IBM WebSphere MQ

Reference

Version 7 Release 5

Note
Before using this information and the product it supports, read the information in "Notices" on page 3781.

This edition applies to version 7 release 5 of WebSphere MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

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

Contents

Figures

Tables

Reference

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

- Syntax diagrams
- "Troubleshooting and support reference" on page 3008

Configuration reference

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

The configuration reference information is provided in the following subtopics:

Related information:

Configuring

Example configuration information

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

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

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

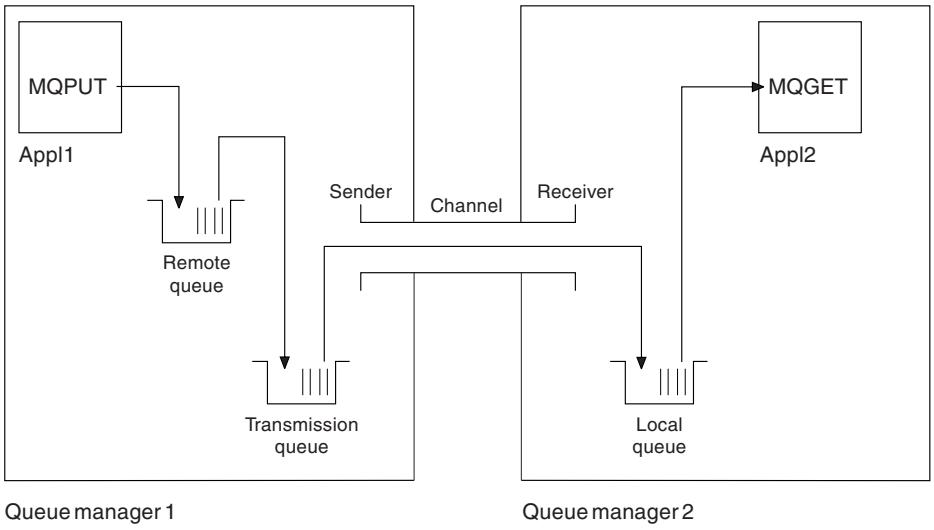


Figure 1. WebSphere MQ channel to be set up in the example configuration

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

The objects in this network are:

- A remote queue
- A transmission queue

- A local queue
- A sender channel
- A receiver channel

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

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

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

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

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

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

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

Network infrastructure in the example

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

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

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

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

Communications software in the example

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

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

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

Related information:

Configuring

How to use the communication examples

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

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

If you are using a Windows, UNIX or Linux system, before you begin to follow the instructions for your platform, you must set various environment variables. Set the environment variables by entering one of the following commands :

- On Windows:

  MQ_INSTALLATION_PATH/bin/setmqenv

  where *MQ_INSTALLATION_PATH* refers to the location where IBM WebSphere MQ is installed.
- On UNIX and Linux systems:

  . *MQ_INSTALLATION_PATH*/bin/setmqenv

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

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

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

There are example configurations for the following platforms:

- "Example configuration - IBM WebSphere MQ for Windows"
- "Example configuration - IBM WebSphere MQ for AIX" on page 13
- "Example configuration - IBM WebSphere MQ for HP-UX" on page 19
- "Example configuration - IBM WebSphere MQ for Solaris" on page 24
- "Example configuration - IBM WebSphere MQ for Linux" on page 29

IT responsibilities

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

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

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

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

Related concepts:

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

Related information:

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

Example configuration - IBM WebSphere MQ for Windows

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

Set up of communication links are shown on the following platforms:

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

When the connection is established, you must define some channels to complete the configuration. Example programs and commands for configuration are described in "WebSphere MQ for Windows configuration" on page 7.

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

Establishing an LU 6.2 connection:

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

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

Establishing a TCP connection:

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

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

```
runmqlsr -t tcp
```

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

What next?

When the TCP/IP connection is established, you are ready to complete the configuration. Go to “WebSphere MQ for Windows configuration” on page 7.

Establishing a NetBIOS connection:

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

To set up a NetBIOS connection, follow these steps:

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

   ```
   NETBIOS:
    LocalName=WNTNETB1
   ```

   and at the receiving end:

   ```
   NETBIOS:
    LocalName=WNTNETB2
   ```

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

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

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

   ```
   SET MQNAME=WNTNETB1I
   ```

This name must be unique.

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

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

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

5. At the receiving end, define the corresponding receiver channel. For example:

```
DEFINE CHANNEL (WINNT.OS2.NET) CHLTYPE(RCVR) +
       TRPTYPE(NETBIOS) +
       REPLACE
```

6. Start the channel initiator because each new channel is started as a thread rather than as a new process.

```
runmqchi
```

7. At the receiving end, start the IBM WebSphere MQ listener:

```
runmqlsr -t netbios
```

   Optionally you can specify values for the queue manager name, NetBIOS local name, number of sessions, number of names, and number of commands. See Defining a NetBIOS connection on Windows for more information about setting up NetBIOS connections.

Establishing an SPX connection:

An SPX connection applies only to a client and server running Windows XP and Windows 2003 Server.

This section contains information about:

- IPX/SPX parameters
- SPX addressing
- Receiving on SPX

IPX/SPX parameters

Refer to the Microsoft documentation for full details of the use and setting of the NWLink IPX and SPX parameters. The IPX/SPX parameters are in the following paths in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NWLinkSPX\Parameters
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NWLinkIPX\Parameters
```

SPX addressing

WebSphere MQ uses the SPX address of each machine to establish connectivity. The SPX address is specified in the following form:

network.node(*socket*)

where

network
: Is the 4-byte network address of the network on which the remote machine resides,

node
: Is the 6-byte node address, which is the LAN address of the LAN adapter in the remote machine

socket
: Is the 2-byte socket number on which the remote machine listens.

The default socket number used by WebSphere MQ is 5E86. You can change the default socket number by specifying it in the Windows registry or in the queue manager configuration file qm.ini. The lines in the Windows registry might read:

```
SPX:
  SOCKET=n
```

For more information about values you can set in qm.ini, see "Configuration file stanzas for distributed queuing" on page 48.

The SPX address is later specified in the CONNAME parameter of the sender channel definition. If the WebSphere MQ systems being connected reside on the same network, the network address need not be specified. Similarly, if the remote system is listening on the default socket number (5E86), it need not be specified. A fully qualified SPX address in the CONNAME parameter is:

```
CONNAME('network.node(socket)')
```

but if the systems reside on the same network and the default socket number is used, the parameter is:

```
CONNAME(node)
```

A detailed example of the channel configuration parameters is given in "WebSphere MQ for Windows configuration."

Receiving on SPX

Receiving channel programs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel.

You should use the WebSphere MQ listener.

Using the WebSphere MQ listener

To run the Listener supplied with WebSphere MQ, that starts new channels as threads, use the RUNMQLSR command. For example:

```
RUNMQLSR -t spx
```

Optionally you can specify the queue manager name or the socket number if you are not using the defaults.

WebSphere MQ for Windows configuration:

Example programs and commands for configuration.

Note:

1. You can use the sample program, AMQSBCG, to show the contents and headers of all the messages in a queue. For example:

   ```
   AMQSBCG q_name qmgr_name
   ```

   shows the contents of the queue *q_name* defined in queue manager *qmgr_name*.

   Alternatively, you can use the message browser in the WebSphere MQ Explorer.
2. You can start any channel from the command prompt using the command

   ```
   runmqchl -c channel.name
   ```
3. Error logs can be found in the directories *MQ_INSTALLATION_PATH*\qmgrs*qmgrname*\errors and *MQ_INSTALLATION_PATH*\qmgrs\@system\errors. In both cases, the most recent messages are at the end of amqerr01.log.

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

4. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Default configuration:

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

For information about using the Postcard application, see Verify the installation using the Postcard application.

Basic configuration:

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

.If you choose the command prompt:

1. Create the queue manager using the command:

   ```
   crtmqm -u dlqname -q winnt
   ```

   where:

   winnt Is the name of the queue manager

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

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

   This command creates a queue manager and a set of default objects.

2. Start the queue manager using the command:

   ```
   strmqm winnt
   ```

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

Channel configuration:

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

The following sections detail the configuration to be performed on the Windows queue manager to implement the channel described in Figure 1 on page 1.

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

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

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

Table 1. Configuration worksheet for WebSphere MQ for Windows

	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**WINNT**	
B	Local queue name		**WINNT.LOCALQ**	
Connection to WebSphere MQ for AIX The values in this section of the table must match those used in Table 2 on page 15, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**WINNT.AIX.SNA**	
H	Sender (TCP) channel name		**WINNT.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.WINNT.SNA**	
J	Receiver (TCP) channel name	H	**AIX.WINNT.TCP**	
Connection to MQSeries® for HP Tru64 UNIX The values in this section of the table must match those used in your HP Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.WINNT.TCP**	
J	Receiver (TCP) channel name	H	**WINNT.DECUX.TCP**	
Connection to WebSphere MQ for HP-UX The values in this section of the table must match those used in Table 3 on page 21, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**WINNT.HPUX.SNA**	
H	Sender (TCP) channel name		**WINNT.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.WINNT.SNA**	
J	Receiver (TCP/IP) channel name	H	**HPUX.WINNT.TCP**	
Connection to WebSphere MQ for Solaris The values in this section of the table must match those used in Table 4 on page 26, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender (SNA) channel name		**WINNT.SOLARIS.SNA**	
H	Sender (TCP) channel name		**WINNT.SOLARIS.TCP**	

Table 1. Configuration worksheet for WebSphere MQ for Windows (continued)

	Parameter Name	Reference	Example Used	User Value
I	Receiver (SNA) channel name	G	**SOLARIS.WINNT.SNA**	
J	Receiver (TCP) channel name	H	**SOLARIS.WINNT.TCP**	
Connection to WebSphere MQ for Linux The values in this section of the table must match those used in Table 5 on page 32, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	
G	Sender (SNA) channel name		**WINNT.LINUX.SNA**	
H	Sender (TCP) channel name		**WINNT.LINUX.TCP**	
I	Receiver (SNA) channel name	G	**LINUX.WINNT.SNA**	
J	Receiver (TCP) channel name	H	**LINUX.WINNT.TCP**	
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**WINNT.AS400.SNA**	
H	Sender (TCP) channel name		**WINNT.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.WINNT.SNA**	
C	Remote queue manager name	A	**MVS™**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**WINNT.MVS.SNA**	
H	Sender (TCP) channel name		**WINNT.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.WINNT.SNA**	
C	Remote queue manager name	A	**QSG**	
D	Remote queue name		**QSG.REMOTEQ**	
E	Queue name at remote system	B	**QSG.SHAREDQ**	
F	Transmission queue name		**QSG**	
G	Sender (SNA) channel name		**WINNT.QSG.SNA**	
H	Sender (TCP) channel name		**WINNT.QSG.TCP**	
I	Receiver (SNA) channel name	G	**QSG.WINNT.SNA**	
Connection to MQSeries for VSE/ESA The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	

Table 1. Configuration worksheet for WebSphere MQ for Windows (continued)

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

WebSphere MQ for Windows sender-channel definitions using SNA:

A code sample.

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

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

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

WebSphere MQ for Windows receiver-channel definitions using SNA:

A code sample.

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

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

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

A code sample.

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

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

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

WebSphere MQ for Windows receiver-channel definitions using TCP:

A code sample.

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

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

Automatic startup:

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

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

For more information, see Administering IBM WebSphere MQ.

Running channels as processes or threads:

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

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

Multiple thread support - pipelining:

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

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

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

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

Note:

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

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

Channel exit considerations

Pipelining can cause some exit programs to fail, because:

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

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

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

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

Example configuration - IBM WebSphere MQ for AIX

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

The following platforms are covered in the examples:

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

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

Establishing an LU 6.2 connection:

Describes the parameters needed for an LU 6.2 connection.

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

Establishing a TCP connection:

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

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

```
runmqlsr -t tcp
```

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

1. Edit the file /etc/services.

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

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

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

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

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

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

What next?

The connection is now established. You are ready to complete the configuration. Go to "WebSphere MQ for AIX configuration."

WebSphere MQ for AIX configuration:

Defining channels to complete the configuration.

Note:

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

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

3. Start any channel using the command:

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

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

Basic configuration

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

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

where:

aix Is the name of the queue manager

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

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

This command creates a queue manager and a set of default objects.

2. Start the queue manager from the AIX command line using the command:

   ```
   strmqm aix
   ```

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

3. Start **runmqsc** from the AIX command line and use it to create the undeliverable message queue by entering the command:

   ```
   def ql (dlqname)
   ```

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

Channel configuration:

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

The following section details the configuration to be performed on the AIX queue manager to implement the channel described in Figure 1 on page 1.

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

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

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

Table 2. Configuration worksheet for WebSphere MQ for AIX

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**AIX**	
B	Local queue name		**AIX.LOCALQ**	
Connection to WebSphere MQ for Windows The values in this section of the table must match those used in Table 1 on page 9, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (SNA) channel name		**AIX.WINNT.SNA**	
H	Sender (TCP/IP) channel name		**AIX.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.AIX.SNA**	
J	Receiver (TCP) channel name	H	**WINNT.AIX.TCP**	
Connection to WebSphere MQ for HP Tru64 UNIX The values in this section of the table must match those used in your HP Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	

Table 2. Configuration worksheet for WebSphere MQ for AIX (continued)

ID	Parameter Name	Reference	Example Used	User Value
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.AIX.TCP**	
J	Receiver (TCP) channel name	H	**AIX.DECUX.TCP**	
Connection to WebSphere MQ for HP-UX The values in this section of the table must match those used in Table 3 on page 21, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**AIX.HPUX.SNA**	
H	Sender (TCP) channel name		**AIX.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.AIX.SNA**	
J	Receiver (TCP) channel name	H	**HPUX.AIX.TCP**	
Connection to WebSphere MQ for Solaris The values in this section of the table must match those used in Table 4 on page 26, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender (SNA) channel name		**AIX.SOLARIS.SNA**	
H	Sender (TCP/IP) channel name		**AIX.SOLARIS.TCP**	
I	Receiver (SNA) channel name	G	**SOLARIS.AIX.SNA**	
J	Receiver (TCP/IP) channel name	H	**SOLARIS.AIX.TCP**	
Connection to WebSphere MQ for Linux The values in this section of the table must match those used in Table 5 on page 32, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	
G	Sender (SNA) channel name		**AIX.LINUX.SNA**	
H	Sender (TCP/IP) channel name		**AIX.LINUX.TCP**	
I	Receiver (SNA) channel name	G	**LINUX.AIX.SNA**	
J	Receiver (TCP/IP) channel name	H	**LINUX.AIX.TCP**	
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	

Table 2. Configuration worksheet for WebSphere MQ for AIX (continued)

ID	Parameter Name	Reference	Example Used	User Value
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**AIX.AS400.SNA**	
H	Sender (TCP) channel name		**AIX.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.AIX.SNA**	
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**AIX.MVS.SNA**	
H	Sender (TCP) channel name		**AIX.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.AIX.SNA**	
C	Remote queue manager name	A	**QSG**	
D	Remote queue name		**QSG.REMOTEQ**	
E	Queue name at remote system	B	**QSG.SHAREDQ**	
F	Transmission queue name		**QSG**	
G	Sender (SNA) channel name		**AIX.QSG.SNA**	
H	Sender (TCP) channel name		**AIX.QSG.TCP**	
I	Receiver (SNA) channel name	G	**QSG.AIX.SNA**	
Connection to MQSeries for VSE/ESA The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**AIX.VSE.SNA**	
I	Receiver channel name	G	**VSE.AIX.SNA**	

WebSphere MQ for AIX sender-channel definitions using SNA:

Example commands.

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

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

def chl (AIX.WINNT.SNA) chltype(sdr) +    G
```

```
    trptype(lu62) +
    conname('WINNTCPIC') +                      17
    xmitq(WINNT) +                              F
    replace
```

WebSphere MQ for AIX receiver-channel definitions using SNA:

Example commands.

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

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

WebSphere MQ for AIX TPN setup:

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

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

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

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

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

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

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

WebSphere MQ for AIX sender-channel definitions using TCP:

Example commands.

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

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

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

WebSphere MQ for AIX receiver-channel definitions using TCP:

Example commands.

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

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

Example configuration - IBM WebSphere MQ for HP-UX

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

The following platforms are included:

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

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

Establishing an LU 6.2 connection:

Describes the parameters needed for an LU 6.2 connection

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

Establishing a TCP connection:

Alternative ways of establishing a connection and next steps.

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

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

1. Edit the file /etc/services.

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

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

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

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

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

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

4. Run the command:
   `kill -1 inetd processid`

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

What next?

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

WebSphere MQ for HP-UX configuration:

Describes defining the channels to complete the configuration.

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

Start any channel using the command:
`runmqchl -c channel.name`

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

Basic configuration
1. Create the queue manager from the UNIX prompt using the command:
   `crtmqm -u dlqname -q hpux`

   where:

   hpux Is the name of the queue manager

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

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

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

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

Channel configuration:

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

The following section details the configuration to be performed on the HP-UX queue manager to implement the channel described in Figure 1 on page 1.

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

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

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

Table 3. Configuration worksheet for WebSphere MQ for HP-UX

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**HPUX**	
B	Local queue name		**HPUX.LOCALQ**	
Connection to WebSphere MQ for Windows The values in this section of the table must match those used in Table 1 on page 9, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (SNA) channel name		**HPUX.WINNT.SNA**	
H	Sender (TCP/IP) channel name		**HPUX.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.HPUX.SNA**	
J	Receiver (TCP) channel name	H	**WINNT.HPUX.TCP**	
Connection to WebSphere MQ for AIX The values in this section of the table must match those used in Table 2 on page 15, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**HPUX.AIX.SNA**	
H	Sender (TCP) channel name		**HPUX.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.HPUX.SNA**	
J	Receiver (TCP) channel name	H	**AIX.HPUX.TCP**	
Connection to WebSphere MQ for HP Tru64 UNIX The values in this section of the table must match those used in your HP Tru64 UNIX system.				

Table 3. Configuration worksheet for WebSphere MQ for HP-UX (continued)

ID	Parameter Name	Reference	Example Used	User Value
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.HPUX.TCP**	
J	Receiver (TCP) channel name	H	**HPUX.DECUX.TCP**	
Connection to WebSphere MQ for Solaris The values in this section of the table must match those used in Table 4 on page 26, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**SOLARIS**	
G	Sender (SNA) channel name		**HPUX.SOLARIS.SNA**	
H	Sender (TCP/IP) channel name		**HPUX.SOLARIS.TCP**	
I	Receiver (SNA) channel name	G	**SOLARIS.HPUX.SNA**	
J	Receiver (TCP/IP) channel name	H	**SOLARIS.HPUX.TCP**	
Connection to WebSphere MQ for Linux The values in this section of the table must match those used in Table 5 on page 32, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	
G	Sender (SNA) channel name		**HPUX.LINUX.SNA**	
H	Sender (TCP/IP) channel name		**HPUX.LINUX.TCP**	
I	Receiver (SNA) channel name	G	**LINUX.HPUX.SNA**	
J	Receiver (TCP/IP) channel name	H	**LINUX.HPUX.TCP**	
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**HPUX.AS400.SNA**	
H	Sender (TCP/IP) channel name		**HPUX.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.HPUX.SNA**	
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	

Table 3. Configuration worksheet for WebSphere MQ for HP-UX (continued)

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

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

Example commands.

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

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

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

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

Example commands.

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

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

WebSphere MQ for HP-UX invokable TP setup:

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

This is not required for HP SNAplus2 Release 6.

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

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

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

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

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

Example commands.

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

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

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

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

Example commands.

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

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

Example configuration - IBM WebSphere MQ for Solaris

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

Examples are given on the following platforms:

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

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

Establishing an LU 6.2 connection using SNAP-IX:

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

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

Establishing a TCP connection:

Information about configuring a TCP connection and next steps.

To establish a TCP connection, follow these steps.

1. Edit the file /etc/services.

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

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

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

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

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

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

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

4. Run the appropriate command, as follows:
   - For Solaris 9:

     ```
     kill -1 inetd processid
     ```

   - For Solaris 10 or later:

     ```
     inetconv
     ```

What next?

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

WebSphere MQ for Solaris configuration:

Describes channels to be defined to complete the configuration.

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

Start any channel using the command:

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

Note:

1. Sample programs are installed in *MQ_INSTALLATION_PATH*/samp.

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

2. Error logs are stored in /var/mqm/qmgrs/*qmgrname*/errors.
3. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

4. For an SNA or LU6.2 channel, if you experience an error when you try to load the communications library, probably file liblu62.so cannot be found. A likely solution to this problem is to add its location, which is probably /opt/SUNWlu62, to LD_LIBRARY_PATH.

Basic configuration

1. Create the queue manager from the UNIX prompt using the command:
   ```
   crtmqm -u dlqname -q solaris
   ```
   where:

   solaris
   Is the name of the queue manager

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

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

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

Channel configuration:

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

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

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

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

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

Table 4. Configuration worksheet for WebSphere MQ for Solaris

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**SOLARIS**	
B	Local queue name		**SOLARIS.LOCALQ**	
Connection to WebSphere MQ for Windows The values in this section of the table must match those used in Table 1 on page 9, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	

Table 4. Configuration worksheet for WebSphere MQ for Solaris (continued)

ID	Parameter Name	Reference	Example Used	User Value
G	Sender (SNA) channel name		**SOLARIS.WINNT.SNA**	
H	Sender (TCP/IP) channel name		**SOLARIS.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.SOLARIS.SNA**	
J	Receiver (TCP) channel name	H	**WINNT.SOLARIS.TCP**	
Connection to WebSphere MQ for AIX The values in this section of the table must match those used in Table 2 on page 15, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**SOLARIS.AIX.SNA**	
H	Sender (TCP) channel name		**SOLARIS.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.SOLARIS.SNA**	
J	Receiver (TCP) channel name	H	**AIX.SOLARIS.TCP**	
Connection to MQSeries for Compaq Tru64 Unix The values in this section of the table must match those used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.SOLARIS.TCP**	
J	Receiver (TCP) channel name	H	**SOLARIS.DECUX.TCP**	
Connection to WebSphere MQ for HP-UX The values in this section of the table must match those used in Table 3 on page 21, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**SOLARIS.HPUX.SNA**	
H	Sender (TCP) channel name		**SOLARIS.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.SOLARIS.SNA**	
J	Receiver (TCP/IP) channel name	H	**HPUX.SOLARIS.TCP**	
Connection to WebSphere MQ for Linux The values in this section of the table must match those used in Table 5 on page 32, as indicated.				
C	Remote queue manager name	A	**LINUX**	
D	Remote queue name		**LINUX.REMOTEQ**	
E	Queue name at remote system	B	**LINUX.LOCALQ**	
F	Transmission queue name		**LINUX**	
G	Sender (SNA) channel name		**SOLARIS.LINUX.SNA**	

Table 4. Configuration worksheet for WebSphere MQ for Solaris (continued)

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

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

Example coding.

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

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

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

WebSphere MQ for Solaris receiver-channel definitions using SNA:

Example coding.

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

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

WebSphere MQ for Solaris sender-channel definitions using TCP:

Example coding.

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

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

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

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

Example coding.

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

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

Example configuration - IBM WebSphere MQ for Linux

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

The examples given are on the following platforms:

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

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

Establishing an LU 6.2 connection:

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

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

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

Establishing a TCP connection on Linux:

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

Using the inet daemon (INETD)

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

To establish a TCP connection, follow these steps.

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

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

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

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

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

   ```
   kill -1 inetd processid
   ```

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

For example:

```
MQSeries1     1414/tcp
MQSeries2     1822/tcp
MQSeries1 stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta -m QM1
MQSeries2 stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta -m QM2
```

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

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

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

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

Using the extended inet daemon (XINETD)

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

To establish a TCP connection, follow these steps.

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

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

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

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

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

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

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

```
cps = 500 60
```

What next?

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

WebSphere MQ for Linux configuration:

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

Start any channel using the command:

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

Note:

1. Sample programs are installed in *MQ_INSTALLATION_PATH*/samp, where *MQ_INSTALLATION_PATH* represents the high-level directory in which WebSphere MQ is installed.

2. Error logs are stored in /var/mqm/qmgrs/*qmgrname*/errors.
3. When you are using the command interpreter **runmqsc** to enter administration commands, a + at the end of a line indicates that the next line is a continuation. Ensure that there is a space between the last parameter and the continuation character.

Basic configuration

1. Create the queue manager from the UNIX prompt using the command:
   ```
   crtmqm -u dlqname -q linux
   ```
   where:

   linux Is the name of the queue manager

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

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

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

Channel configuration:

The following section details the configuration to be performed on the Linux queue manager to implement the channel described in Figure 1 on page 1.

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

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

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

Table 5. Configuration worksheet for WebSphere MQ for Linux

ID	Parameter Name	Reference	Example Used	User Value
Definition for local node				
A	Queue Manager Name		**LINUX**	
B	Local queue name		**LINUX.LOCALQ**	
Connection to WebSphere MQ for Windows The values in this section of the table must match those used in Table 1 on page 9, as indicated.				
C	Remote queue manager name	A	**WINNT**	
D	Remote queue name		**WINNT.REMOTEQ**	
E	Queue name at remote system	B	**WINNT.LOCALQ**	
F	Transmission queue name		**WINNT**	
G	Sender (SNA) channel name		**LINUX.WINNT.SNA**	

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

ID	Parameter Name	Reference	Example Used	User Value
H	Sender (TCP/IP) channel name		**LINUX.WINNT.TCP**	
I	Receiver (SNA) channel name	G	**WINNT.LINUX.SNA**	
J	Receiver (TCP) channel name	H	**WINNT.LINUX.TCP**	
Connection to WebSphere MQ for AIX The values in this section of the table must match those used in Table 2 on page 15, as indicated.				
C	Remote queue manager name	A	**AIX**	
D	Remote queue name		**AIX.REMOTEQ**	
E	Queue name at remote system	B	**AIX.LOCALQ**	
F	Transmission queue name		**AIX**	
G	Sender (SNA) channel name		**LINUX.AIX.SNA**	
H	Sender (TCP) channel name		**LINUX.AIX.TCP**	
I	Receiver (SNA) channel name	G	**AIX.LINUX.SNA**	
J	Receiver (TCP) channel name	H	**AIX.LINUX.TCP**	
Connection to MQSeries for Compaq Tru64 UNIX The values in this section of the table must match those used in your Compaq Tru64 UNIX system.				
C	Remote queue manager name	A	**DECUX**	
D	Remote queue name		**DECUX.REMOTEQ**	
E	Queue name at remote system	B	**DECUX.LOCALQ**	
F	Transmission queue name		**DECUX**	
H	Sender (TCP) channel name		**DECUX.LINUX.TCP**	
J	Receiver (TCP) channel name	H	**LINUX.DECUX.TCP**	
Connection to WebSphere MQ for HP-UX The values in this section of the table must match those used in Table 3 on page 21, as indicated.				
C	Remote queue manager name	A	**HPUX**	
D	Remote queue name		**HPUX.REMOTEQ**	
E	Queue name at remote system	B	**HPUX.LOCALQ**	
F	Transmission queue name		**HPUX**	
G	Sender (SNA) channel name		**LINUX.HPUX.SNA**	
H	Sender (TCP) channel name		**LINUX.HPUX.TCP**	
I	Receiver (SNA) channel name	G	**HPUX.LINUX.SNA**	
J	Receiver (TCP/IP) channel name	H	**HPUX.LINUX.TCP**	
Connection to WebSphere MQ for Solaris The values in this section of the table must match those used in Table 4 on page 26, as indicated.				
C	Remote queue manager name	A	**SOLARIS**	
D	Remote queue name		**SOLARIS.REMOTEQ**	
E	Queue name at remote system	B	**SOLARIS.LOCALQ**	
F	Transmission queue name		**GIS**	
G	Sender (SNA) channel name		**LINUX.SOLARIS.SNA**	
H	Sender (TCP/IP) channel name		**LINUX.SOLARIS.TCP**	

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

ID	Parameter Name	Reference	Example Used	User Value
I	Receiver (SNA) channel name	G	**SOLARIS.LINUX.SNA**	
J	Receiver (TCP/IP) channel name	H	**SOLARIS.LINUX.TCP**	
C	Remote queue manager name	A	**AS400**	
D	Remote queue name		**AS400.REMOTEQ**	
E	Queue name at remote system	B	**AS400.LOCALQ**	
F	Transmission queue name		**AS400**	
G	Sender (SNA) channel name		**LINUX.AS400.SNA**	
H	Sender (TCP) channel name		**LINUX.AS400.TCP**	
I	Receiver (SNA) channel name	G	**AS400.LINUX.SNA**	
C	Remote queue manager name	A	**MVS**	
D	Remote queue name		**MVS.REMOTEQ**	
E	Queue name at remote system	B	**MVS.LOCALQ**	
F	Transmission queue name		**MVS**	
G	Sender (SNA) channel name		**LINUX.MVS.SNA**	
H	Sender (TCP) channel name		**LINUX.MVS.TCP**	
I	Receiver (SNA) channel name	G	**MVS.LINUX.SNA**	
Connection to MQSeries for VSE/ESA (WebSphere MQ for Linux (x86 platform) only) The values in this section of the table must match those used in your VSE/ESA system.				
C	Remote queue manager name	A	**VSE**	
D	Remote queue name		**VSE.REMOTEQ**	
E	Queue name at remote system	B	**VSE.LOCALQ**	
F	Transmission queue name		**VSE**	
G	Sender channel name		**LINUX.VSE.SNA**	
I	Receiver channel name	G	**VSE.LINUX.SNA**	

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

Example coding.

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

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

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

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

Example coding.

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

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

WebSphere MQ for Linux sender-channel definitions using TCP:

Example coding.

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

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

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

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

Example coding.

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

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

Queue names

Use this information to understand the restrictions of queue names and reserved queue names.

Queues can have names up to 48 characters long.

Reserved Queue names

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

Queue Name	Description
SYSTEM.ADMIN.ACTIVITY.QUEUE	Queue for activity reports
SYSTEM.ADMIN.CHANNEL.EVENT	Queue for channel events
SYSTEM.ADMIN.COMMAND.EVENT	Queue for command events
SYSTEM.ADMIN.COMMAND.QUEUE	Queue to which PCF command messages are sent
SYSTEM.ADMIN.CONFIG.EVENT	Queue for configuration events
SYSTEM.ADMIN.PERFM.EVENT	Queue for performance events
SYSTEM.ADMIN.PUBSUB.EVENT	System publish/subscribe related event queue

Queue Name	Description
SYSTEM.ADMIN.QMGR.EVENT	Queue for queue manager events
SYSTEM.ADMIN.TRACE.ROUTE.QUEUE	Queue for trace-route reply messages
SYSTEM.AUTH.DATA.QUEUE	The queue that holds access control lists for the queue manager. (Not for z/OS)
SYSTEM.CHANNEL.INITQ	Initiation queue for channels
SYSTEM.CHANNEL.SYNCQ	The queue that holds the synchronization data for channels
SYSTEM.CHLAUTH.DATA.QUEUE	IBM WebSphere MQ channel authentication data queue
SYSTEM.CICS.INITIATION.QUEUE	Queue used for triggering (not for z/OS)
SYSTEM.CLUSTER.COMMAND.QUEUE	Queue used to communicate repository changes between queue managers (AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS only)
SYSTEM.CLUSTER.HISTORY.QUEUE	The queue is used to store the history of cluster state information for service purposes.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	Queue used to hold information about the repository (AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS only)
SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE	The queue is used to create individual transmit queues for each cluster-sender channel.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	Transmission queue for all destinations managed by cluster support (AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS only)
SYSTEM.COMMAND.INPUT	Queue to which command messages are sent on z/OS
SYSTEM.COMMAND.REPLY.MODEL	Model queue definition for command replies (for z/OS)
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter queue (not for z/OS)
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue definition
SYSTEM.DEFAULT.INITIATION.QUEUE	Queue used to trigger a specified process (not for z/OS)
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue definition
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue definition
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue definition
SYSTEM.DURABLE.SUBSCRIBER.QUEUE	A local queue used to hold a persistent copy of the durable subscriptions in the queue manager
SYSTEM.HIERARCHY.STATE	Queue used to hold information about the state of inter-queue manager relationships in a publish/subscribe hierarchy
SYSTEM.JMS.TEMPQ.MODEL	Model for JMS temporary queues
SYSTEM.INTERNAL.REPLY.QUEUE	IBM WebSphere MQ internal reply queue (not for z/OS)
SYSTEM.INTER.QMGR.CONTROL	Queue used in a publish/subscribe hierarchy to receive requests from a remote queue manager to create a proxy subscription
SYSTEM.INTER.QMGR.PUBS	Queue used in a publish/subscribe hierarchy to receive publications from a remote queue manager
SYSTEM.INTER.QMGR.FANREQ	Queue used in a publish/subscribe hierarchy to process requests to create a proxy subscription on a remote queue manager
SYSTEM.MQEXPLORER.REPLY.MODEL	Model queue definition for replies for IBM WebSphere MQ Explorer
SYSTEM.MQSC.REPLY.QUEUE	Model queue definition for MQSC command replies (not for z/OS)

Queue Name	Description
SYSTEM.QSG.CHANNEL.SYNCQ	Shared local queue used for storing messages that contain the synchronization information for shared channels (z/OS only)
SYSTEM.QSG.TRANSMIT.QUEUE	Shared local queue used by the intra-group queuing agent when transmitting messages between queue managers in the same queue-sharing group (z/OS only)
SYSTEM.RETAINED.PUB.QUEUE	A local queue used to hold a copy of each retained publication in the queue manager.
SYSTEM.SELECTION.EVALUATION.QUEUE	IBM WebSphere MQ internal selection evaluation queue (not for z/OS)
SYSTEM.SELECTION.VALIDATION.QUEUE	IBM WebSphere MQ internal selection validation queue (not for z/OS)

Other object names

Processes, namelists, clusters, topics, services, and authentication information objects can have names up to 48 characters long. Channels can have names up to 20 characters long. Storage classes can have names up to 8 characters long. CF structures can have names up to 12 characters long.

Reserved object names

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

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

Object Name	Description
SYSTEM.DEFAULT.LISTENER.SPX	Default SPX listener (Windows only)
SYSTEM.DEFAULT.LISTENER.TCP	Default TCP/IP listener (IBM i, Windows, UNIX and Linux systems only)
SYSTEM.DEFAULT.NAMELIST	Default namelist definition
SYSTEM.DEFAULT.PROCESS	Default process definition
SYSTEM.DEFAULT.SEVICE	Default service (IBM i, Windows, UNIX and Linux systems only)
SYSTEM.DEFAULT.TOPIC	Default topic definition
SYSTEM.QPUBSUB.QUEUE.NAMELIST	A list of queues for the Queued Publish/Subscribe interface to monitor
SYSTEMST	Default storage class definition (z/OS only)

Queue name resolution

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

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

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

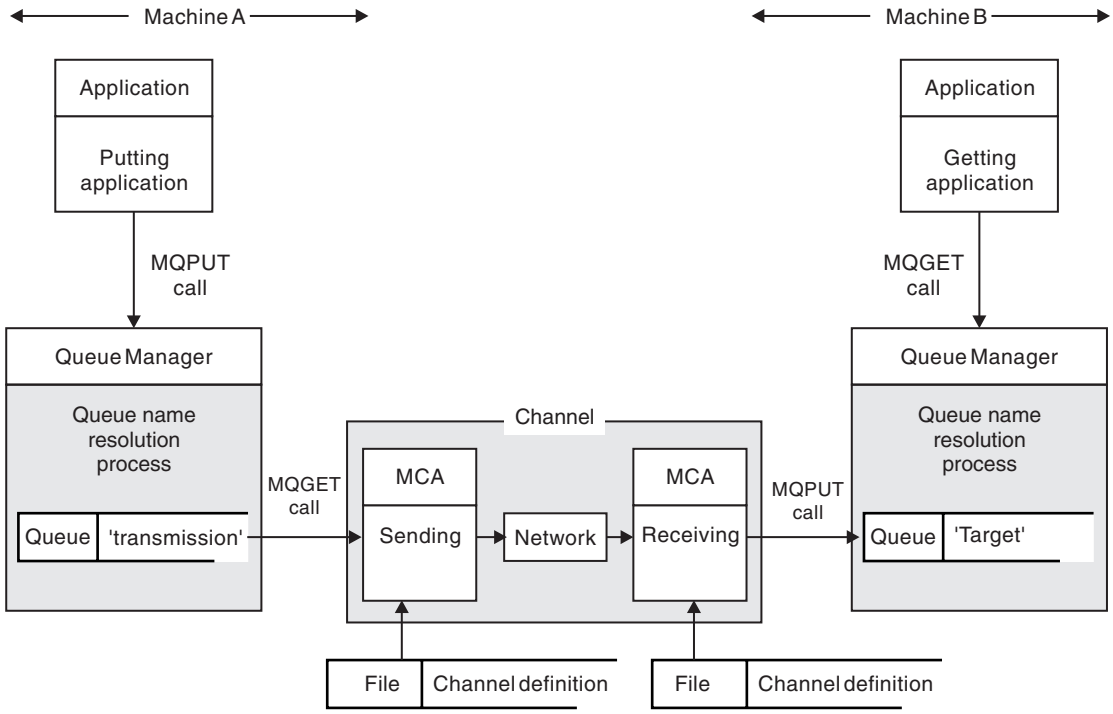


Figure 2. Name resolution

Referring to Figure 2, the basic mechanism for putting messages on a remote queue, as far as the application is concerned, is the same as for putting messages on a local queue:

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

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

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

The sequence of events is as follows:

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

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

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

What is queue name resolution?

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

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

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

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

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

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

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

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

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

System and default objects

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

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

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

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

- Table 6 lists the system and default queue objects.
- Table 7 on page 42 lists the system and default topic objects.
- Table 8 on page 42 lists the system and default channel objects.
- Table 9 on page 43 lists the system and default authentication information objects.
- Table 10 on page 43 lists the system and default listener objects.
- Table 11 on page 43 lists the system and default namelist objects.
- Table 12 on page 43 lists the system and default process objects.
- Table 13 on page 43 lists the system and default service objects.

Table 6. System and default objects: queues

Object name	Description
SYSTEM.ADMIN.ACCOUNTING.QUEUE	The queue that holds accounting monitoring data.
SYSTEM.ADMIN.ACTIVITY.QUEUE	The queue that holds returned activity reports.
SYSTEM.ADMIN.CHANNEL.EVENT	Event queue for channels.
SYSTEM.ADMIN.COMMAND.EVENT	Event queue for command events.
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue. Used for remote MQSC commands and PCF commands.
SYSTEM.ADMIN.CONFIG.EVENT	Event queue for configuration events.
SYSTEM.ADMIN.PERFM.EVENT	Event queue for performance events.
SYSTEM.ADMIN.PUBSUB.EVENT	System publish/subscribe related event queue
SYSTEM.ADMIN.QMGR.EVENT	Event queue for queue manager events.
SYSTEM.ADMIN.STATISTICS.QUEUE	The queue that holds statistics monitoring data.
SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE	The queue that displays trace activity.
SYSTEM.ADMIN.TRACE.ROUTE.QUEUE	The queue that holds returned trace-route reply messages.
SYSTEM.AUTH.DATA.QUEUE	The queue that holds access control lists for the queue manager.
SYSTEM.CHANNEL.INITQ	Channel initiation queue.
SYSTEM.CHANNEL.SYNCQ	The queue that holds the synchronization data for channels.
SYSTEM.CHLAUTH.DATA.QUEUE	IBM WebSphere MQ channel authentication data queue
SYSTEM.CICS.INITIATION.QUEUE	Default CICS initiation queue.
SYSTEM.CLUSTER.COMMAND.QUEUE	The queue used to carry messages to the repository queue manager.
SYSTEM.CLUSTER.HISTORY.QUEUE	The queue is used to store the history of cluster state information for service purposes.

Table 6. System and default objects: queues (continued)

Object name	Description
SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE	The queue is used to create individual transmit queues for each cluster-sender channel.
SYSTEM.CLUSTER.REPOSITORY.QUEUE	The queue used to store all repository information.
SYSTEM.CLUSTER.TRANSMIT.QUEUE	The transmission queue for all messages to all clusters.
SYSTEM.DEAD.LETTER.QUEUE	Dead-letter (undelivered-message) queue.
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue.
SYSTEM.DEFAULT.INITIATION.QUEUE	Default initiation queue.
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue.
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue.
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue.
SYSTEM.JMS.TEMPQ.MODEL	Model for JMS temporary queues
SYSTEM.MQEXPLORER.REPLY.MODEL	The IBM WebSphere MQ Explorer reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to the IBM WebSphere MQ Explorer.
SYSTEM.MQSC.REPLY.QUEUE	MQSC command reply-to queue. This is a model queue that creates a temporary dynamic queue for replies to remote MQSC commands.
SYSTEM.PENDING.DATA.QUEUE	Support deferred messages in JMS.

Table 7. System and default objects: topics

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

Table 8. System and default objects: channels

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

Table 9. System and default objects: authentication information objects

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

Table 10. System and default objects: listeners

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

1. Windows only

Table 11. System and default objects: namelists

Object name	Description
SYSTEM.DEFAULT.NAMELIST	Default namelist.

Table 12. System and default objects: processes

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

Table 13. System and default objects: services

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

Windows default configuration objects

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

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

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

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

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

The cluster name for the Windows default configuration is DEFAULT_CLUSTER.

If the queue manager is not a repository queue manager, the objects listed in Table 14 are created.

Table 14. Objects created by the Windows default configuration application

Object	Name
Queue manager	The short TCP/IP name prefixed with the characters QM_. The maximum length of the queue manager name is 48 characters. Names exceeding this limit are truncated at 48 characters. If the last character of the name is a period (.), this is replaced by a space ( ). The queue manager has a command server, a channel listener, and channel initiator associated with it. The channel listener listens on the standard WebSphere MQ port, port number 1414. Any other queue managers created on this machine must not use port 1414 while the default configuration queue manager still exists.
Generic cluster receiver channel	The short TCP/IP name prefixed with the characters TO_QM_. The maximum length of the generic cluster receiver name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ( ).
Cluster sender channel	The cluster sender channel is initially created with the name TO_+QMNAME+. Once WebSphere MQ has established a connection to the repository queue manager for the default configuration cluster, this name is replaced with the name of the repository queue manager for the default configuration cluster, prefixed with the characters TO_. The maximum length of the cluster sender channel name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ( ).
Local message queue	The local message queue is called `default`.
Local message queue for use by the WebSphere MQ Postcard application	The local message queue for use by the WebSphere MQ Postcard application is called `postcard`.
Server connection channel	The server connection channel allows clients to connect to the queue manager. Its name is the short TCP/IP name, prefixed with the characters S_. The maximum length of the server connection channel name is 20 characters. Names exceeding this limit are truncated at 20 characters. If the last character of the name is a period (.), this is replaced by a space ( ).

If the queue manager is a repository queue manager, the default configuration is similar to that described in Table 14, but with the following differences:

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

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

SYSTEM.BASE.TOPIC

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

Table 15. Default values of SYSTEM.BASE.TOPIC

Parameter	Value
TOPICSTR	"
CLUSTER	The default value is an empty string.
COMMINFO	SYSTEM.DEFAULT.COMMINFO.MULTICAST
DEFPRESP	SYNC
DEFPRTY	0
DEFPSIST	NO
DESCR	'Base topic for resolving attributes'
DURSUB	YES
MCAST	DISABLED
MDURMDL	SYSTEM.DURABLE.MODEL.QUEUE
MNDURMDL	SYSTEM.NDURABLE.MODEL.QUEUE
NPMSGDLV	ALLAVAIL
PMSGDLV	ALLDUR
PROXYSUB	FIRSTUSE
PUB	ENABLED
PUBSCOPE	ALL
SUB	ENABLED
SUBSCOPE	ALL
USEDLQ	YES
WILDCARD	PASSTHRU

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

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

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

Stanza information

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

Configuring stanzas

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

- Changing IBM WebSphere MQ configuration information helps you configure the:
  - *AllQueueManagers* stanza
  - *DefaultQueueManager* stanza
  - *ExitProperties* stanza
  - *LogDefaults* stanza
  - *Security* stanza in the `qm.ini` file
- Changing queue manager configuration information helps you configure the:
  - *AccessMode* stanza (Windows only)
  - *Service* stanza - for Installable services
  - *Log* stanza
  - *RestrictedMode* stanza (UNIX and Linux systems only)
  - *XAResourceManager* stanza
  - *TCP*, *LU62*, and *NETBIOS* stanzas
  - *ExitPath* stanza
  - *QMErrorLog* stanza
  - *SSL* stanza
  - *ExitPropertiesLocal* stanza
- Configuring services and components helps you configure the:
  - *Service* stanza
  - *ServiceComponent* stanza

  and contains links to how they are used for different services on UNIX and Linux, and Windows platforms.
- Configuring API exits helps you configure the:
  - *AllActivityTrace* stanza
  - *AppplicationTrace* stanza
- Configuring activity trace behavior helps you configure the:
  - *ApiExitCommon* stanza
  - *ApiExitTemplate* stanza
  - *ApiExitLocal* stanza
- Configuration information for clients helps you configure the:
  - *CHANNELS* stanza
  - *ClientExitPath* stanza
  - *LU62*, *NETBIOS* and *SPX* stanza (Windows only)
  - *MessageBuffer* stanza
  - *SSL* stanza
  - *TCP* stanza
- "Configuration file stanzas for distributed queuing" on page 48 helps you configure the:
  - *CHANNELS* stanza
  - *TCP* stanza

  - *LU62* stanza
  - *NETBIOS*
  - *ExitPath* stanza
- Setting queued publish/subscribe message attributes helps you configure the:
  - *PersistentPublishRetry* attribute
  - *NonPersistentPublishRetry* attribute
  - *PublishBatchSize* attribute
  - *PublishRetryInterval* attribute

  in the *Broker* stanza.

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

Configuration files

See:

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

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

mqs.ini file

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

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

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

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

qm.ini file

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

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

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

  To configure *InstallableServices*:
  - On UNIX and Linux platforms, use the *Service* and *ServiceComponent* stanzas.
  - On Windows, use **regedit**.
- *Connection* for *DefaultBindType*

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

- XAResourceManager

In addition, you can use the crtmqm command to change these properties:

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

mqclient.ini file

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

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

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

Configuration file stanzas for distributed queuing

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

This topic shows the stanzas in the queue manager configuration file that relate to distributed queuing. It applies to the queue manager configuration file for IBM WebSphere MQ on Windows, UNIX and Linux systems. The file is called qm.ini on all platforms.

The stanzas that relate to distributed queuing are:

- CHANNELS
- TCP
- LU62
- NETBIOS
- SPX (Windows XP and Windows 2003 Server only)
- EXITPATH

Figure 3 on page 49 shows the values that you can set using these stanzas. When you are defining one of these stanzas, you do not need to start each item on a new line. You can use either a semicolon (;) or a hash character (#) to indicate a comment.

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

Figure 3. qm.ini stanzas for distributed queuing

Note:

1. MQIBINDTYPE applies only to IBM WebSphere MQ for AIX, IBM WebSphere MQ for HP-UX, and IBM WebSphere MQ for Solaris.
2. EXITPATH applies only to IBM WebSphere MQ for AIX, IBM WebSphere MQ for HP-UX, and IBM WebSphere MQ for Solaris.

Related information:

Configuring

Changing configuration information on Windows, UNIX, and Linux systems

Channel attributes

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

This information is product-sensitive programming interface information.

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

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

Note: In WebSphere MQ for IBM i, most attributes can be specified as `*SYSDFTCHL`, which means that the value is taken from the system default channel in your system.

Channel attributes and channel types

Different types of channel support different channel attributes.

The channel types for WebSphere MQ channel attributes are listed in Table 16.

Table 16. Channel attributes for the channel types

Attribute field	MQSC command parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
Alter date	ALTDATE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Alter time	ALTTIME	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Batch heartbeat interval	BATCHHB	Yes	Yes					Yes	Yes
Batch interval	BATCHINT	Yes	Yes					Yes	Yes
Batch size	BATCHSZ	Yes	Yes	Yes	Yes			Yes	Yes
Channel name	CHANNEL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Channel statistics	STATCHL	Yes	Yes	Yes	Yes			Yes	Yes
Channel type	CHLTYPE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Client channel weight	CLNTWGHT					Yes			
Cluster	CLUSTER							Yes	Yes
Cluster namelist	CLUSNL							Yes	Yes
Cluster workload priority	CLWLPRTY							Yes	Yes
Cluster workload rank	CLWLRANK							Yes	Yes
Cluster workload weight	CLWLWGHT							Yes	Yes
Connection affinity	AFFINITY					Yes			
Connection name	CONNAME	Yes	Yes		Yes	Yes		Yes	Yes
Convert message	CONVERT	Yes	Yes					Yes	Yes
Data compression	COMPMSG	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Description	DESCR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Disconnect interval	DISCINT	Yes	Yes					Yes	Yes
Header compression	COMPHDR	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Heartbeat interval	HBINT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Keepalive interval	KAINT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Local address	LOCLADDR	Yes	Yes		Yes	Yes		Yes	Yes
Long retry count	LONGRTY	Yes	Yes					Yes	Yes
Long retry interval	LONGTMR	Yes	Yes					Yes	Yes
LU 6.2 mode name	MODENAME	Yes	Yes		Yes	Yes		Yes	Yes
LU 6.2 transaction program name	TPNAME	Yes	Yes		Yes	Yes		Yes	Yes
Maximum instances	MAXINST						Yes		
Maximum instances per client	MAXINSTC						Yes		
Maximum message length	MAXMSGL	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 16. Channel attributes for the channel types (continued)

Attribute field	MQSC command parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
Message channel agent name	MCANAME	Yes	Yes		Yes			Yes	Yes
Message channel agent type	MCATYPE	Yes	Yes		Yes			Yes	Yes
Message channel agent user	MCAUSER	Yes	Yes	Yes	Yes		Yes	Yes	Yes
Message exit name	MSGEXIT	Yes	Yes	Yes	Yes			Yes	Yes
Message exit user data	MSGDATA	Yes	Yes	Yes	Yes			Yes	Yes
Message-retry exit name	MREXIT			Yes	Yes				Yes
Message-retry exit user data	MRDATA			Yes	Yes				Yes
Message retry count	MRRTY			Yes	Yes				Yes
Message retry interval	MRTMR			Yes	Yes				Yes
Monitoring	MONCHL	Yes	Yes	Yes	Yes		Yes	Yes	Yes
Network-connection priority	NETPRTY								Yes
Nonpersistent message speed	NPMSPEED	Yes	Yes	Yes	Yes			Yes	Yes
Password	PASSWORD	Yes	Yes		Yes	Yes		Yes	
Property control	PROPCTL	Yes	Yes					Yes	Yes
PUT authority	PUTAUT			Yes	Yes		Yes[1]		Yes
Queue manager name	QMNAME					Yes			
Receive exit name	RCVEXIT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Receive exit user data	RCVDATA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Security exit name	SCYEXIT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Security exit user data	SCYDATA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Send exit name	SENDEXIT	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Send exit user data	SENDDATA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Sequence number wrap	SEQWRAP	Yes	Yes	Yes	Yes			Yes	Yes
Shared connections	SHARECNV					Yes	Yes		
Short retry count	SHORTRTY	Yes	Yes					Yes	Yes
Short retry interval	SHORTTMR	Yes	Yes					Yes	Yes
SSL Cipher Specification	SSLCIPH	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
SSL Client Authentication	SSLCAUTH		Yes	Yes	Yes		Yes		Yes
SSL Peer	SSLPEER	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Transmission queue name	XMITQ	Yes	Yes						
Transport type	TRPTYPE	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Table 16. Channel attributes for the channel types (continued)

Attribute field	MQSC command parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
Use Dead-Letter Queue	USEDLQ	Yes	Yes	Yes	Yes			Yes	Yes
User ID	USERID	Yes	Yes		Yes	Yes		Yes	

Related concepts:

"Channel attributes in alphabetical order"
This section describes each attribute of a channel object, with its valid values and notes on its use where appropriate.

Related information:

MQSC reference
Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

Channel attributes in alphabetical order

This section describes each attribute of a channel object, with its valid values and notes on its use where appropriate.

WebSphere MQ for some platforms might not implement all the attributes shown in this section. Exceptions and platform differences are mentioned in the individual attribute descriptions, where relevant.

The keyword that you can specify in MQSC is shown in brackets for each attribute.

The attributes are arranged in alphabetical order.

Alter date (ALTDATE):

This attribute is the date on which the definition was last altered, in the form `yyyy-mm-dd`.

This attribute is valid for all channel types.

Alter time (ALTTIME):

This attribute is the time at which the definition was last altered, in the form `hh:mm:ss`.

This attribute is valid for all channel types.

Batch Heartbeat Interval (BATCHHB):

This attribute allows a sending channel to verify that the receiving channel is still active just before committing a batch of messages.

The batch heartbeat interval thus allows the batch to be backed out rather than becoming in-doubt if the receiving channel is not active. By backing out the batch, the messages remain available for processing so they could, for example, be redirected to another channel.

If the sending channel has had a communication from the receiving channel within the batch heartbeat interval, the receiving channel is assumed to be still active, otherwise a 'heartbeat' is sent to the receiving channel to check.

The value is in milliseconds and must be in the range zero through 999999. A value of zero indicates that batch heart beating is not used.

This attribute is valid for channel types of:
- Sender
- Server
- Cluster sender
- Cluster receiver

Batch interval (BATCHINT):

This attribute is a period, in milliseconds, during which the channel keeps a batch open even if there are no messages on the transmission queue.

You can specify any number of milliseconds, from zero through 999 999 999. The default value is zero.

If you do not specify a batch interval, the batch closes when the number of messages specified in BATCHSZ has been sent or when the transmission queue becomes empty. On lightly loaded channels, where the transmission queue frequently becomes empty the effective batch size might be much smaller than BATCHSZ.

You can use the BATCHINT attribute to make your channels more efficient by reducing the number of short batches. Be aware, however, that you can slow down the response time, because batches last longer and messages remain uncommitted for longer.

If you specify a BATCHINT, batches close only when one of the following conditions is met:
- The number of messages specified in BATCHSZ have been sent.
- There are no more messages on the transmission queue and a time interval of BATCHINT has elapsed while waiting for messages (since the first message of the batch was retrieved).

Note: BATCHINT specifies the total amount of time that is spent waiting for messages. It does not include the time spent retrieving messages that are already available on the transmission queue, or the time spent transferring messages.

This attribute is valid for channel types of:
- Sender
- Server
- Cluster sender
- Cluster receiver

Batch size (BATCHSZ):

This attribute is the maximum number of messages to be sent before a sync point is taken.

The batch size does not affect the way the channel transfers messages; messages are always transferred individually, but are committed or backed out as a batch.

To improve performance, you can set a batch size to define the maximum number of messages to be transferred between two *sync points*. The batch size to be used is negotiated when a channel starts, and the lower of the two channel definitions is taken. On some implementations, the batch size is calculated from the lowest of the two channel definitions and the two queue manager MAXUMSGS values. The actual size of a batch can be less; for example, a batch completes when there are no messages left on the transmission queue or the batch interval expires.

A large value for the batch size increases throughput, but recovery times are increased because there are more messages to back out and send again. The default BATCHSZ is 50, and you are advised to try that value first. You might choose a lower value for BATCHSZ if your communications are unreliable, making the need to recover more likely.

Sync point procedure needs a unique logical unit of work identifier to be exchanged across the link every time a sync point is taken, to coordinate batch commit procedures.

If the synchronized batch commit procedure is interrupted, an *in-doubt* situation might arise. In-doubt situations are resolved automatically when a message channel starts. If this resolution is not successful, manual intervention might be necessary, using the RESOLVE command.

Some considerations when choosing the number for batch size:

- If the number is too large, the amount of queue space taken up on both ends of the link becomes excessive. Messages take up queue space when they are not committed, and cannot be removed from queues until they are committed.
- If there is likely to be a steady flow of messages, you can improve the performance of a channel by increasing the batch size because fewer confirm flows are needed to transfer the same quantity of bytes.
- If message flow characteristics indicate that messages arrive intermittently, a batch size of 1 with a relatively large disconnect time interval might provide a better performance.
- The number can be in the range 1 through 9999. However, for data integrity reasons, channels connecting to any of the current platforms must specify a batch size greater than 1. A value of 1 is for use with Version 1 products, apart from WebSphere MQ for MVS.
- Even though nonpersistent messages on a fast channel do not wait for a sync point, they do contribute to the batch-size count.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Channel name (CHANNEL):

This attribute specifies the name of the channel definition.

The name can contain up to 20 characters, although as both ends of a message channel must have the same name, and other implementations might have restrictions on the size, the actual number of characters might have to be smaller.

Where possible, channel names are unique to one channel between any two queue managers in a network of interconnected queue managers.

The name must contain characters from the following list:

Alphabetic	(A-Z, a-z; note that uppercase and lowercase are significant)
Numerics	(0-9)
Period	(.)
Forward slash	(/)
Underscore	(_)
Percentage sign	(%)

Note:

1. Embedded blanks are not allowed, and leading blanks are ignored.
2. On systems using EBCDIC Katakana, you cannot use lowercase characters.

This attribute is valid for all channel types.

Channel statistics (STATCHL):

This attribute controls the collection of statistics data for channels.

The possible values are:

QMGR
Statistics data collection for this channel is based upon the setting of the queue manager attribute STATCHL. This value is the default value.

OFF Statistics data collection for this channel is disabled.

LOW Statistics data collection for this channel is enabled with a low ratio of data collection.

MEDIUM
Statistics data collection for this channel is enabled with a moderate ratio of data collection.

HIGH Statistics data collection for this channel is enabled with a high ratio of data collection.

For more information about channel statistics, see Monitoring reference.

This attribute is not supported on z/OS.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Channel type (CHLTYPE):

This attribute specifies the type of the channel being defined.

The possible channel types are:

Message channel types:

- Sender
- Server
- Receiver
- Requester

- Cluster-sender
- Cluster-receiver

MQI channel types:

- Client-connection (WebSphere MQ for Windows systems, and UNIX systems only)

  Note: Client-connection channels can also be defined on z/OS for use on other platforms.
- Server-connection

The two ends of a channel must have the same name and have compatible types:

- Sender with receiver
- Requester with server
- Requester with sender (for callback)
- Server with receiver (server is used as a sender)
- Client-connection with server-connection
- Cluster-sender with cluster-receiver

Client channel weight (CLNTWGHT):

This attribute specifies a weighting to influence which client-connection channel definition is used.

The client channel weighting attribute is used so that client channel definitions can be selected at random based on their weighting when more than one suitable definition is available.

When a client issues an MQCONN requesting connection to a queue manager group, by specifying a queue manager name starting with an asterisk, which enables client weight balancing across several queue managers, and more than one suitable channel definition is available in the client channel definition table (CCDT), the definition to use is randomly selected based on the weighting, with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

Specify a value in the range 0 - 99. The default is 0.

A value of 0 indicates that no load balancing is performed and applicable definitions are selected in alphabetical order. To enable load balancing choose a value in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest. The distribution of connections between two or more channels with non-zero weightings is proportional to the ratio of those weightings. For example, three channels with CLNTWGHT values of 2, 4, and 14 are selected approximately 10%, 20%, and 70% of the time. This distribution is not guaranteed. If the AFFINITY attribute of the connection is set to PREFERRED, the first connection chooses a channel definition according to client weightings, and then subsequent connections continue to use the same channel definition.

This attribute is valid for the client-connection channel type only.

Cluster (CLUSTER):

This attribute is the name of the cluster to which the channel belongs.

The maximum length is 48 characters conforming to the rules for naming WebSphere MQ objects.

Up to one of the resultant values of CLUSTER or CLUSNL can be non-blank. If one of the values is non-blank, the other must be blank.

This attribute is valid for channel types of:

- Cluster sender

- Cluster receiver

Cluster namelist (CLUSNL):

This attribute is the name of the namelist that specifies a list of clusters to which the channel belongs.

Up to one of the resultant values of CLUSTER or CLUSNL can be nonblank. If one of the values is nonblank, the other must be blank.

This attribute is valid for channel types of:
- Cluster sender
- Cluster receiver

Cluster workload priority (CLWLPRTY):

This attribute specifies the priority of the channel.

The value must be in the range 0 through 9, where 0 is the lowest priority and 9 is the highest.

This attribute is valid for channel types of:
- Cluster sender
- Cluster receiver

Cluster workload rank (CLWLRANK):

This attribute specifies the rank of the channel.

The value must be in the range 0 through 9, where 0 is the lowest rank and 9 is the highest.

This attribute is valid for channel types of:
- Cluster sender
- Cluster receiver

Cluster workload weight (CLWLWGHT):

This attribute applies a weighting factor to the channel so the proportion of messages sent down that channel can be controlled.

The value must be in the range 1 through 99, where 1 is the lowest weighting and 99 is the highest.

This attribute is valid for channel types of:
- Cluster sender
- Cluster receiver

Connection affinity (AFFINITY):

This attribute specifies whether client applications that connect multiple times using the same queue manager name, use the same client channel.

Use this attribute when multiple applicable channel definitions are available.

The possible values are:

PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the client channel weight, with any definitions having a weight of 0 first and in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful definitions with client channel weight values other than 0 are moved to the end of the list. Definitions with a client channel weight of 0 remain at the start of the list and are selected first for each connection.

Each client process with the same host name always creates the same list.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET), and for applications that use the IBM WebSphere MQ classes for Java™ and IBM WebSphere MQ classes for JMS, the list is updated if the CCDT has been modified since the list was created.

This value is the default value.

NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the client channel weight, with any definitions having a weight of 0 selected first in alphabetical order.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET), and for applications that use the IBM WebSphere MQ classes for Java and IBM WebSphere MQ classes for JMS, the list is updated if the CCDT has been modified since the list was created.

This attribute is valid for the client-connection channel type only.

Connection name (CONNAME):

This attribute is the communications connection identifier. It specifies the particular communications links to be used by this channel.

It is optional for server channels, unless the server channel is triggered, in which case it must specify a connection name.

Specify CONNAME as a comma-separated list of names of machines for the stated TRPTYPE. Typically only one machine name is required. You can provide multiple machine names to configure multiple connections with the same properties. The connections are usually tried in the order they are specified in the connection list until a connection is successfully established. The order is modified for clients if the CLNTWGHT attribute is provided. If no connection is successful, the channel attempts the connection again, as determined by the attributes of the channel. With client channels, a connection-list provides an alternative to using queue manager groups to configure multiple connections. With message channels, a connection list is used to configure connections to the alternative addresses of a multi-instance queue manager.

Providing multiple connection names in a list was first supported in IBM WebSphere MQ Version 7.0.1. It changes the syntax of the CONNAME parameter. Earlier clients and queue managers connect using the

first connection name in the list, and do not read the rest of the connection names in the list. In order for the earlier clients and queue managers to parse the new syntax, you must specify a port number on the first connection name in the list. Specifying a port number avoids problems when connecting to the channel from a client or queue manager that is running at a level earlier than IBM WebSphere MQ Version 7.0.1.

On AIX, HP-UX, IBM i, Linux, Solaris, and Windows platforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, WebSphere MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example: `(1415)`

The generated `CONNAME` is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

The name is up to 48 characters (see note 1) for z/OS, 264 characters for other platforms, and:

If the transport type is TCP

CONNAME is either the host name or the network address of the remote machine (or the local machine for cluster-receiver channels). For example, `(ABC.EXAMPLE.COM)`, `(2001:DB8:0:0:0:0:0:0)` or `(127.0.0.1)`. It can include the port number, for example `(MACHINE(123))`. It can include the IP_name of a z/OS dynamic DNS group or a Network Dispatcher input port.

If you use an IPV6 address in a network that only supports IPV4, the connection name is not resolved. In a network which uses both IPV4 and IPV6, Connection name interacts with Local Address to determine which IP stack is used. See “Local Address (LOCLADDR)” on page 65 for further information.

If the transport type is LU 6.2

For WebSphere MQ for IBM i, Windows systems, and UNIX systems, give the fully-qualified name of the partner LU if the TPNAME and MODENAME are specified. For other versions or if the TPNAME and MODENAME are blank, give the CPI-C side information object name for your specific platform.

On z/OS, there are two forms in which to specify the value:

- Logical unit name

  The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. This name can be specified in one of three forms:

Form	Example
luname	`IGY12355`
luname/TPname	`IGY12345/APING`
luname/TPname/modename	`IGY12345/APINGD/#INTER`

  For the first form, the TP name and mode name must be specified for the TPNAME and MODENAME attributes; otherwise these attributes must be blank.

  Note: For client-connection channels, only the first form is allowed.

- Symbolic name

  The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The TPNAME and MODENAME attributes must be blank.

Note: For cluster-receiver channels, the side information is on the other queue managers in the cluster. Alternatively, in this case it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

The specified or implied LU name can be that of a VTAM generic resources group.

If the transmission protocol is NetBIOS

CONNAME is the NetBIOS name defined on the remote machine.

If the transmission protocol is SPX

CONNAME is an SPX-style address consisting of a 4 byte network address, a 6 byte node address and a 2 byte socket number. Enter these values in hexadecimal, with the network and node addresses separated by a period and the socket number in brackets. For example:

```
CONNAME('0a0b0c0d.804abcde23a1(5e86)')
```

If the socket number is omitted, the default WebSphere MQ SPX socket number is used. The default is X'5E86'.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

It is optional for server channels, unless the server channel is triggered, in which case it must specify a connection name.

Note:

1. A workaround to the 48 character limit might be one of the following suggestions:
   - Set up your DNS servers so that you use, for example, host name of "myserver" instead of "myserver.location.company.com", ensuring you can use the short host name.
   - Use IP addresses.
2. The definition of transmission protocol is contained in “Transport type (TRPTYPE)” on page 82.

Convert message (CONVERT):

This attribute specifies that the message must be converted into the format required by the receiving system before transmission.

Application message data is typically converted by the receiving application. However, if the remote queue manager is on a platform that does not support data conversion, use this channel attribute to specify that the message must be converted into the format required by the receiving system ***before*** transmission.

The possible values are `yes` and `no`. If you specify `yes`, the application data in the message is converted before sending if you have specified one of the built-in format names, or a data conversion exit is available for a user-defined format (See Writing data-conversion exits). If you specify `no`, the application data in the message is not converted before sending.

This attribute is valid for channel types of:

- Sender
- Server

- Cluster sender
- Cluster receiver

Data compression (COMPMSG):

This attribute is a list of message data compression techniques supported by the channel.

For sender, server, cluster-sender, cluster-receiver, and client-connection channels the values specified are in order of preference. The first compression technique supported by the remote end of the channel is used. The channels' mutually supported compression techniques are passed to the sending channel's message exit where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits. See "Header compression (COMPHDR)" on page 63 for compression of the message header.

The possible values are:

NONE
No message data compression is performed. This value is the default value.

RLE Message data compression is performed using run-length encoding.

ZLIBFAST
Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

ZLIBHIGH
Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

ANY Allows the channel to support any compression technique that the queue manager supports. Only supported for Receiver, Requester and Server-Connection channels.

This attribute is valid for all channel types.

Description (DESCR):

This attribute describes the channel definition and contains up to 64 bytes of text.

Note: The maximum number of characters is reduced if the system is using a double byte character set (DBCS).

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager to ensure that the text is translated correctly if it is sent to another queue manager.

This attribute is valid for all channel types.

Disconnect interval (DISCINT):

This attribute is the length of time after which a channel closes down, if no message arrives during that period.

This attribute is a time-out attribute, specified in seconds, for the server, cluster-sender, sender, and cluster-receiver channels. The interval is measured from the point at which a batch ends, that is when the batch size is reached or when the batch interval expires and the transmission queue becomes empty. If no messages arrive on the transmission queue during the specified time interval, the channel closes down. (The time is approximate.)

The close-down exchange of control data between the two ends of the channel includes an indication of the reason for closing. This ensures that the corresponding end of the channel remains available to start again.

You can specify any number of seconds from zero through 999 999 where a value of zero means no disconnect; wait indefinitely.

For server-connection channels using the TCP protocol, the interval represents the client inactivity disconnect value, specified in seconds. If a server-connection has received no communication from its partner client for this duration, it terminates the connection. The server-connection inactivity interval applies under the following circumstances:

- between WebSphere MQ API calls from a client
- between an MQGET call, only if:
  - client application executes an MQGET with WaitInterval
  - the DISCINT parameter is set on the server-connection channel and is smaller than the MQGET WaitInterval
  - the SHARECNV parameter of the server-connection channel is greater than 0.

This attribute is valid for channel types of:

- Sender
- Server
- Server connection
- Cluster sender
- Cluster receiver

This attribute is not applicable for server-connection channels using protocols other than TCP.

Note: Performance is affected by the value specified for the disconnect interval.

A low value (for example a few seconds) can be detrimental to system performance by constantly starting the channel. A large value (more than an hour) might mean that system resources are needlessly held up. You can also specify a heartbeat interval, so that when there are no messages on the transmission queue, the sending MCA sends a heartbeat flow to the receiving MCA, thus giving the receiving MCA an opportunity to quiesce the channel without waiting for the disconnect interval to expire. For these two values to work together effectively, the heartbeat interval value must be significantly lower than the disconnect interval value.

The default DISCINT value is set to 100 minutes. However, a value of a few minutes is often a reasonable value to use without impacting performance or keeping channels running for unnecessarily long periods of time. If it is appropriate for your environment you can change this value, either on each individual channel or through changing the value in the default channel definitions, for example SYSTEM.DEF.SENDER.

For more information, see Stopping and quiescing channels.

Disposition (QSGDISP):

This attribute specifies the disposition of the channel in a queue-sharing group. It is valid on z/OS only.

Values are:

QMGR
The channel is defined on the page set of the queue manager that executes the command. This value is the default.

GROUP
The channel is defined in the shared repository. This value is allowed only if there is a shared queue manager environment. When a channel is defined with QSGDISP(GROUP), the command DEFINE CHANNEL(name) NOREPLACE QSGDISP(COPY) is generated automatically and sent to all active queue managers to cause them to make local copies on page set 0. For queue managers which are not active, or which join the queue sharing group at a later date, the command is generated when the queue manager starts.

COPY The channel is defined on the page set of the queue manager that executes the command, copying its definition from the QSGDISP(GROUP) channel of the same name. This value is allowed only if there is a shared queue manager environment.

This attribute is valid for all channel types.

Header compression (COMPHDR):

This attribute is a list of header data compression techniques supported by the channel.

For sender, server, cluster-sender, cluster-receiver, and client-connection channels the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used. The channels' mutually supported compression techniques are passed to the sending channel's message exit where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

Possible values are:

NONE
No header data compression is performed. This value is the default value.

SYSTEM
Header data compression is performed.

This attribute is valid for all channel types.

Heartbeat interval (HBINT):

This attribute specifies the approximate time between heartbeat flows that are to be passed from a sending message channel agent (MCA) when there are no messages on the transmission queue.

Heartbeat flows unblock the receiving MCA, which is waiting for messages to arrive or for the disconnect interval to expire. When the receiving MCA is unblocked, it can disconnect the channel without waiting for the disconnect interval to expire. Heartbeat flows also free any storage buffers that have been allocated for large messages and close any queues that have been left open at the receiving end of the channel.

The value is in seconds and must be in the range 0 - 999 999. A value of zero means that no heartbeat flows are to be sent. The default value is 300. To be most useful, the value must be significantly less than the disconnect interval value.

With applications that use IBM WebSphere MQ classes for Java, JMS or .NET APIs, the HBINT value is determined in one of the following ways:

- Either by the value on the SVRCONN channel that is used by the application.
- Or by the value on the CLNTCONN channel, if the application has been configured to use a CCDT.

For server-connection and client-connection channels, heartbeats can flow from both the server side as well as the client side independently. If no data has been transferred across the channel for the heartbeat interval, the client-connection MQI agent sends a heartbeat flow and the server-connection MQI agent responds to it with another heartbeat flow. This happens irrespective of the state of the channel, for example, irrespective of whether it is inactive while making an API call, or is inactive waiting for client user input. The server-connection MQI agent is also capable of initiating a heartbeat to the client, again irrespective of the state of the channel. To prevent both server-connection and client-connection MQI agents heart beating to each other at the same time, the server heartbeat is flowed after no data has been transferred across the channel for the heartbeat interval plus 5 seconds.

For server-connection and client-connection channels working in the channel mode before IBM WebSphere MQ Version 7.0, heartbeats flow only when a server MCA is waiting for an MQGET command with the WAIT option specified, which it has issued on behalf of a client application.

For more information about making MQI channels work in the two modes, see SharingConversations (MQLONG).

Related information:

DEFINE CHANNEL
Use the MQSC command **DEFINE CHANNEL** to define a new channel, and set its parameters.

ALTER CHANNEL
Use the MQSC command ALTER CHANNEL to alter the parameters of a channel.

Keepalive Interval (KAINT):

This attribute is used to specify a timeout value for a channel.

The Keepalive Interval attribute is a value passed to the communications stack specifying the Keepalive timing for the channel. It allows you to specify a different keepalive value for each channel.

You can set the Keepalive Interval (KAINT) attribute for channels on a per-channel basis. On platforms other than z/OS, you can access and modify the parameter, but it is only stored and forwarded; there is no functional implementation of the parameter. If you need the functionality provided by the KAINT parameter, use the Heartbeat Interval (HBINT) parameter, as described in “Heartbeat interval (HBINT)” on page 63.

For this attribute to have any effect, TCP/IP keepalive must be enabled. On z/OS, you do enable keepalive by issuing the ALTER QMGR TCPKEEP(YES) MQSC command. On other platforms, it occurs when the KEEPALIVE=YES parameter is specified in the TCP stanza in the distributed queuing configuration file, qm.ini, or through the IBM WebSphere MQ Explorer. Keepalive must also be switched on within TCP/IP itself, using the TCP profile configuration data set.

The value indicates a time, in seconds, and must be in the range 0 - 99999. A Keepalive Interval value of 0 indicates that channel-specific Keepalive is not enabled for the channel and only the system-wide Keepalive value set in TCP/IP is used. You can also set KAINT to a value of AUTO (this value is the default). If KAINT is set to AUTO, the Keepalive value is based on the value of the negotiated heartbeat interval (HBINT) as follows:

Table 17. Negotiated HBINT value and the corresponding KAINT value

Negotiated HBINT	KAINT
>0	Negotiated HBINT + 60 seconds
0	0

If AUTO is specified for KAINT, and it is a server-connection channel, the TCP INTERVAL value is used instead for the keepalive interval.

This attribute is valid for all channel types.

The value is ignored for all channels that have a TransportType (TRPTYPE) other than TCP or SPX

Local Address (LOCLADDR):

This attribute specifies the local communications address for the channel.

This attribute only applies if the transport type (TRPTYPE) is TCP/IP. For all other transport types, it is ignored.

When a LOCLADDR value is specified, a channel that is stopped and then restarted continues to use the TCP/IP address specified in LOCLADDR. In recovery scenarios, this attribute might be useful when the channel is communicating through a firewall. It is useful because it removes problems caused by the channel restarting with the IP address of the TCP/IP stack to which it is connected. LOCLADDR can also force a channel to use an IPv4 or IPv6 stack on a dual stack system, or a dual-mode stack on a single stack system.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

When LOCLADDR includes a network address, the address must be a network addresses belonging to a network interface on the system where the channel is run. For example, when defining a sender channel on queue manager ALPHA to queue manager BETA with the following MSQC command:

```
DEFINE CHANNEL(TO.BETA) CHLTYPE(SDR) CONNAME(192.0.2.0) XMITQ(BETA) LOCLADDR(192.0.2.1)
```

The LOCLADDR address is the IPv4 address `192.0.2.1`. This sender channel runs on the system of queue manager ALPHA, so the IPv4 address must belong to one of the network interfaces its system.

The value is the optional IP address, and optional port or port range used for outbound TCP/IP communications. The format for this information is as follows:

Table 73 on page 452 shows how the LOCLADDR parameter can be used:
`LOCLADDR([ip-addr][(low-port[,high-port])]`*`[,[ip-addr][(low-port[,high-port])])]`*`)`

The maximum length of LOCLADDR, including multiple addresses, is `MQ_LOCAL_ADDRESS_LENGTH`.

If you omit LOCLADDR, a local address is automatically allocated.

Note, that you can set LOCLADDR for a C client using the Client Channel Definition Table (CCDT).

All the parameters are optional. Omitting the `ip-addr` part of the address is useful to enable the configuration of a fixed port number for an IP firewall. Omitting the port number is useful to select a particular network adapter without having the identify a unique local port number. The TCP/IP stack generates a unique port number.

Specify *`[,[ip-addr][(low-port[,high-port])]]`* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *`[,[ip-addr][(low-port[,high-port])]]`* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

`ip-addr`
`ip-addr` is specified in one of three forms:

IPv4 dotted decimal
For example `192.0.2.1`

IPv6 hexadecimal notation
For example `2001:DB8:0:0:0:0:0:0`

Alphanumeric host name form
For example `WWW.EXAMPLE.COM`

`low-port and high-port`
`low-port` and `high-port` are port numbers enclosed in parentheses.

Table 18. Examples of how the LOCLADDR parameter can be used

LOCLADDR	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98, 9.20.4.99	Channel binds to either IP address. The address might be two network adapters on one server, or a different network adapter on two different servers in a multi-instance configuration.
9.20.4.98(1000)	Channel binds to this address and port 1000 locally
9.20.4.98(1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to port in range 1000 - 2000 locally

When a channel is started the values specified for connection name (CONNAME) and local address (LOCLADDR) determine which IP stack is used for communication. The IP stack used is determined as follows:

- If the system has only an IPv4 stack configured, the IPv4 stack is always used. If a local address (LOCLADDR) or connection name (CONNAME) is specified as an IPv6 network address, an error is generated and the channel fails to start.
- If the system has only an IPv6 stack configured, the IPv6 stack is always used. If a local address (LOCLADDR) is specified as an IPv4 network address, an error is generated and the channel fails to start. On platforms supporting IPv6 mapped addressing, if a connection name (CONNAME) is specified as an IPv4 network address, the address is mapped to an IPv6 address. For example, `xxx.xxx.xxx.xxx` is mapped to `::ffff:xxx.xxx.xxx.xxx`. The use of mapped addresses might require protocol translators. Avoid the use of mapped addresses where possible.

- If a local address (LOCLADDR) is specified as an IP address for a channel, the stack for that IP address is used. If the local address (LOCLADDR) is specified as a host name resolving to both IPv4 and IPv6 addresses, the connection name (CONNAME) determines which of the stacks is used. If both the local address (LOCLADDR) and connection name (CONNAME) are specified as host names resolving to both IPv4 and IPv6 addresses, the stack used is determined by the queue manager attribute IPADDRV.
- If the system has dual IPv4 and IPv6 stacks configured and a local address (LOCLADDR) is not specified for a channel, the connection name (CONNAME) specified for the channel determines which IP stack to use. If the connection name (CONNAME) is specified as a host name resolving to both IPv4 and IPv6 addresses, the stack used is determined by the queue manager attribute IPADDRV.

On distributed platforms, it is possible to set a default local address value that will be used for all sender channels that do not have a local address defined. The default value is defined by setting the MQ_LCLADDR environment variable prior to starting the queue manager. The format of the value matches that of MQSC attribute LOCLADDR.

Local addresses with cluster sender channels

Cluster sender channels always inherit the configuration of the corresponding cluster receiver channel as defined on the target queue manager. This is true even if there is a locally defined cluster sender channel of the same name, in which case the manual definition is only used for initial communication.

For this reason, it is not possible to depend on the LOCLADDR defined in the cluster receiver channel as it is likely that the IP address is not owned by the system where the cluster senders are created. For this reason, the LOCLADDR on the cluster receiver should not be used unless there is a reason to restrict only the ports, but not the IP address, for all potential cluster senders, and it is known that those ports are available on all systems where a cluster sender channel may be created.

If a cluster must use LOCLADDR to get the outbound communication channels to bind to a specific IP address, either use a Channel Auto-Definition Exit, or use the default LOCLADDR for the queue manager when possible. When using a channel exit, it forces the LOCLADDR value from the exit into any of the automatically defined CLUSSDR channels.

If using a non-default LOCLADDR for cluster sender channels through the use of an exit or a default value, any matching manually defined cluster sender channel, for example to a full repository queue manager, must also have the LOCLADDR value set to enable initial communication over the channel.

Note: If the operating system returns a bind error for the port supplied in LOCLADDR (or all ports, if a port range is supplied), the channel does not start; the system issues an error message.

Related information:

Working with auto-defined cluster-sender channels

Long retry count (LONGRTY):

This attribute specifies the maximum number of times that the channel is to try allocating a session to its partner.

If the initial allocation attempt fails, the *short retry count* number is decremented and the channel retries the remaining number of times. If it still fails, it retries a *long retry count* number of times with an interval of *long retry interval* between each try. If it is still unsuccessful, the channel closes down. The channel must then be restarted with a command (it is not started automatically by the channel initiator).

(Retry is not attempted if the cause of failure is such that a retry is not likely to be successful.)

If the channel initiator (on z/OS) or the channel (on distributed platforms) is stopped while the channel is retrying, the *short retry count* and *long retry count* are reset when the channel initiator or the channel is restarted, or when a message is successfully put at the sender channel. However, if the channel initiator (on z/OS) or queue manager (on distributed platforms) is shut down and restarted, the *short retry count* and *long retry count* are not reset. The channel retains the retry count values it had before the queue manager restart or the message being put.

Note: For IBM i, UNIX systems, and Windows systems:

1. When a channel goes from RETRYING state to RUNNING state, the *short retry count* and *long retry count* are not reset immediately. They are reset only once the first message flows across the channel successfully after the channel went into RUNNING state, that is; once the local channel confirms the number of messages sent to the other end.
2. The *short retry count* and *long retry count* are reset when the channel is restarted.

The *long retry count* attribute can be set from zero through 999 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

Note: For UNIX systems, and Windows systems, in order for retry to be attempted a channel initiator must be running. The channel initiator must be monitoring the initiation queue specified in the definition of the transmission queue that the channel is using.

Long retry interval (LONGTMR):

This attribute is the approximate interval in seconds that the channel is to wait before retrying to establish connection, during the long retry mode.

The interval between retries can be extended if the channel has to wait to become active.

The channel tries to connect *long retry count* number of times at this long interval, after trying the *short retry count* number of times at the short retry interval.

This attribute can be set from zero through 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

LU 6.2 mode name (MODENAME):

This attribute is for use with LU 6.2 connections. It gives extra definition for the session characteristics of the connection when a communication session allocation is performed.

When using side information for SNA communications, the mode name is defined in the CPI-C Communications Side Object or APPC side information, and this attribute must be left blank; otherwise, it must be set to the SNA mode name.

The name must be one to eight alphanumeric characters long.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

It is not valid for receiver or server-connection channels.

LU 6.2 transaction program name (TPNAME):

This attribute is for use with LU 6.2 connections. It is the name, or generic name, of the transaction program (MCA) to be run at the far end of the link.

When using side information for SNA communications, the transaction program name is defined in the CPI-C Communications Side Object or APPC side information and this attribute must be left blank. Otherwise, this name is required by sender channels and requester channels.

The name can be up to 64 characters long.

The name must be set to the SNA transaction program name, unless the CONNAME contains a side-object name in which case it must be set to blanks. The actual name is taken instead from the CPI-C Communications Side Object, or the APPC side information data set.

This information is set in different ways on different platforms; see Connecting applications using distributed queuing for more information about setting up communication for your platform.

This attribute is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender
- Cluster receiver

Maximum instances (MAXINST):

This attribute specifies the maximum number of simultaneous instances of a server-connection channel that can be started.

This attribute can be set from zero through 999 999 999. A value of zero indicates that no client connections are allowed on this channel. The default value is 999 999 999.

The Client Attachment feature (CAF) is an option of WebSphere MQ for z/OS that supports the attachment of clients to z/OS. If you do not have the Client Attachment feature (CAF) installed, the attribute can be set from zero to five only on the SYSTEM.ADMIN.SVRCONN channel. A value greater than five is interpreted as zero without the CAF installed.

If the value is reduced below the number of instances of the server-connection channel that are currently running, then the running channels are not affected. However, new instances are not able to start until sufficient existing ones have ceased to run.

This attribute is valid for server-connection channels only.

Maximum instances per client (MAXINSTC):

This attribute specifies the maximum number of simultaneous instances of a server-connection channel that can be started from a single client.

This attribute can be set from zero through 999 999 999. A value of zero indicates that no client connections are allowed on this channel. The default value is 999 999 999.

The Client Attachment feature (CAF) is an option of WebSphere MQ for z/OS that supports the attachment of clients to z/OS. If you do not have the Client Attachment feature (CAF) installed, the attribute can be set from zero to five only on the SYSTEM.ADMIN.SVRCONN channel. A value greater than five is interpreted as zero without the CAF installed.

If the value is reduced below the number of instances of the server-connection channel that are currently running from individual clients, then the running channels are not affected. However, new instances from those clients are not able to start until sufficient existing ones have ceased to run.

This attribute is valid for server-connection channels only.

Maximum message length (MAXMSGL):

This attribute specifies the maximum length of a message that can be transmitted on the channel.

On WebSphere MQ for IBM i, UNIX systems, and Windows systems, specify a value greater than or equal to zero, and less than or equal to the maximum message length for the queue manager. See the MAXMSGL parameter of the ALTER QMGR command in ALTER QMGR for more information. On WebSphere MQ for z/OS, specify a value greater than or equal to zero, and less than or equal to 104 857 600 bytes.

Because various implementations of WebSphere MQ systems exist on different platforms, the size available for message processing might be limited in some applications. This number must reflect a size that your system can handle without stress. When a channel starts, the lower of the two numbers at each end of the channel is taken.

By adding the digital signature and key to the message, WebSphere MQ Advanced Message Security increases the length of the message.

Note:

1. You can use a maximum message size of 0 which is taken to mean that the size is to be set to the local queue manager maximum value.

This attribute is valid for all channel types.

Message channel agent name (MCANAME):

This attribute is reserved and if specified must only be set to blanks.

Its maximum length is 20 characters.

Message channel agent type (MCATYPE):

This attribute can specify the message channel agent as a *process* or a *thread*.

On WebSphere MQ for z/OS, it is supported only for channels with a channel type of cluster-receiver.

Advantages of running as a process include:

- Isolation for each channel providing greater integrity
- Job authority specific for each channel
- Control over job scheduling

Advantages of threads include:

- Much reduced use of storage
- Easier configuration by typing on the command line
- Faster execution - it is quicker to start a thread than to instruct the operating system to start a process

For channel types of sender, server, and requester, the default is `process`. For channel types of cluster-sender and cluster-receiver, the default is `thread`. These defaults can change during your installation.

If you specify `process` on the channel definition, a RUNMQCHL process is started. If you specify `thread`, the MCA runs on a thread of the AMQRMPPA process, or of the RUNMQCHI process if MQNOREMPOOL is specified. On the machine that receives the inbound allocates, the MCA runs as a thread if you use RUNMSLSR. It runs as a process if you use **`inetd`**.

On WebSphere MQ for z/OS, this attribute is supported only for channels with a channel type of cluster-receiver. On other platforms, it is valid for channel types of:

- Sender
- Server
- Requester
- Cluster sender
- Cluster receiver

Message channel agent user identifier (MCAUSER):

This attribute is the user identifier (a string) to be used by the MCA for authorization to access IBM WebSphere MQ resources.

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL).

This authorization includes (if PUT authority is DEF) putting the message to the destination queue for receiver or requester channels.

On IBM WebSphere MQ for Windows, the user identifier can be domain-qualified by using the format, `user@domain`, where the `domain` must be either the Windows systems domain of the local system, or a trusted domain.

If this attribute is blank, the MCA uses its default user identifier. For more information, see DEFINE CHANNEL.

This attribute is valid for channel types of:

- Receiver
- Requester
- Server connection
- Cluster receiver

Related information:

Channel authentication records

Message exit name (MSGEXIT):

This attribute specifies the name of the user exit program to be run by the channel message exit.

This attribute can be a list of names of programs that are to be run in succession. Leave blank, if no channel message exit is in effect.

The format and maximum length of this attribute depend on the platform, as for "Receive exit name (RCVEXIT)" on page 77.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Message exit user data (MSGDATA):

This attribute specifies user data that is passed to the channel message exits.

You can run a sequence of message exits. The limitations on the user data length and an example of how to specify MSGDATA for more than one exit are as shown for RCVDATA. See "Receive exit user data (RCVDATA)" on page 77.

This attribute is valid for channel types of:
- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Message-retry exit name (MREXIT):

This attribute specifies the name of the user exit program to be run by the message-retry user exit.

Leave blank if no message-retry exit program is in effect.

The format and maximum length of the name depend on the platform, as for "Receive exit name (RCVEXIT)" on page 77. However, there can only be one message-retry exit specified

This attribute is valid for channel types of:
- Receiver
- Requester
- Cluster receiver

Message-retry exit user data (MRDATA):

This attribute specifies data passed to the channel message-retry exit when it is called.

This attribute is valid for channel types of:
- Receiver
- Requester
- Cluster receiver

Message retry count (MRRTY):

This attribute specifies the number of times the channel tries to redeliver the message.

This attribute controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRRTY is passed to the exit, but the number of attempts made (if any) is controlled by the exit, and not by this attribute.

The value must be in the range 0 - 999 999 999. A value of zero means that no additional attempts are made. The default is 10.

This attribute is valid for channel types of:
- Receiver
- Requester

- Cluster receiver

Message retry interval (MRTMR):

This attribute specifies the minimum interval of time that must pass before the channel can retry the MQPUT operation.

This time interval is in milliseconds.

This attribute controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRTMR is passed to the exit for use by the exit, but the retry interval is controlled by the exit, and not by this attribute.

The value must be in the range 0 - 999 999 999. A value of zero means that the retry is performed as soon as possible (if the value of MRRTY is greater than zero). The default is 1000.

This attribute is valid for the following channel types:
- Receiver
- Requester
- Cluster receiver

Monitoring (MONCHL):

This attribute controls the collection of online Monitoring data.

Possible values are:

QMGR
The collection of Online Monitoring Data is inherited from the setting of the MONCHL attribute in the queue manager object. This value is the default value.

OFF Online Monitoring Data collection for this channel is switched off.

LOW A low ratio of data collection with a minimal effect on performance. However, the monitoring results shown might not be up to date.

MEDIUM
A moderate ratio of data collection with limited effect on the performance of the system.

HIGH A high ratio of data collection with the possibility of an effect on performance. However, the monitoring results shown are the most current.

This attribute is valid for channel types of:
- Sender
- Server
- Receiver
- Requester
- Server connection
- Cluster sender
- Cluster receiver

For more information about monitoring data, see Displaying queue and channel monitoring data.

Network-connection priority (NETPRTY):

This attribute specifies the priority for the network connection.

Distributed queuing chooses the path with the highest priority if there are multiple paths available. The value must be in the range 0 through 9; 0 is the lowest priority.

This attribute is valid for channel types of:

- Cluster receiver

Nonpersistent message speed (NPMSPEED):

This attribute specifies the speed at which nonpersistent messages are to be sent.

Possible values are:

NORMAL
Nonpersistent messages on a channel are transferred within transactions.

FAST Nonpersistent messages on a channel are not transferred within transactions.

The default is FAST. The advantage of this is that nonpersistent messages become available for retrieval far more quickly. The disadvantage is that because they are not part of a transaction, messages might be lost if there is a transmission failure or if the channel stops when the messages are in transit. See Safety of messages.

This attribute is valid for channel types of:

- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Password (PASSWORD):

This attribute specifies a password that can be used by the MCA when attempting to initiate a secure LU 6.2 session with a remote MCA.

You can specify a password of maximum length 12 characters, although only the first 10 characters are used.

It is valid for channel types of sender, server, requester, or client-connection.

On WebSphere MQ for z/OS, this attribute is valid only for client connection channels. On other platforms, it is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender

PUT authority (PUTAUT):

This attribute specifies the type of security processing to be carried out by the MCA.

This attribute is valid for channel types of:

- Receiver
- Requester
- Cluster receiver

Use this attribute to choose the type of security processing to be carried out by the MCA when executing:

- An MQPUT command to the destination queue (for message channels), or
- An MQI call (for MQI channels).

You can choose one of the following:

Process security, also called default authority (DEF)
The default user ID is used.

On all platforms, the user ID used to check open authority on the queue is that of the process or user running the MCA at the receiving end of the message channel.

The queues are opened with this user ID and the open option MQOO_SET_ALL_CONTEXT.

Context security (CTX)
The user ID from the context information associated with the message is used as an alternate user ID.

The *UserIdentifier* in the message descriptor is moved into the *AlternateUserId* field in the object descriptor. The queue is opened with the open options MQOO_SET_ALL_CONTEXT and MQOO_ALTERNATE_USER_AUTHORITY.

On all platforms, the user ID used to check open authority on the queue for MQOO_SET_ALL_CONTEXT and MQOO_ALTERNATE_USER_AUTHORITY is that of the process or user running the MCA at the receiving end of the message channel. The user ID used to check open authority on the queue for MQOO_OUTPUT is the *UserIdentifier* in the message descriptor.

Context security (CTX) is not supported on server-connection channels.

Further details about context fields and open options can be found in Controlling context information.

More information about security can be found in:

- Security
- Setting up security on Windows, UNIX and Linux systems for WebSphere MQ UNIX systems and Windows systems,

Queue manager name (QMNAME):

This attribute specifies the name of the queue manager or queue manager group to which a WebSphere MQ MQI client application can request connection.

This attribute is valid for channel types of:

- Client connection

Receive exit name (RCVEXIT):

This attribute specifies the name of the user exit program to be run by the channel receive user exit.

This attribute can be a list of names of programs that are to be run in succession. Leave blank, if no channel receive user exit is in effect.

The format and maximum length of this attribute depend on the platform:

- On z/OS it is a load module name, maximum length 8 characters, except for client-connection channels where the maximum length is 128 characters.
- On IBM i, it is of the form:
  libname/progname

  when specified in CL commands.

  When specified in WebSphere MQ Commands (MQSC) it has the form:
  progname libname

  where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary). The maximum length of the string is 20 characters.
- On Windows, it is of the form:
  dllname(*functionname*)

  where *dllname* is specified without the suffix `.DLL`. The maximum length of the string is 40 characters.
- On UNIX systems, it is of the form:
  libraryname(*functionname*)

  The maximum length of the string is 40 characters.

During cluster sender channel auto-definition on z/OS, channel exit names are converted to z/OS format. If you want to control how exit names are converted, you can write a channel auto-definition exit. For more information, see Channel auto-definition exit program.

You can specify a list of receive, send, or message exit program names. The names must be separated by a comma, a space, or both. For example:

```
RCVEXIT(exit1 exit2)
MSGEXIT(exit1,exit2)
SENDEXIT(exit1, exit2)
```

The total length of the string of exit names and strings of user data for a particular type of exit is limited to 500 characters. In WebSphere MQ for IBM i, you can list up to 10 exit names. In WebSphere MQ for z/OS, you can list up to eight exit names.

This attribute is valid for all channel types.

Receive exit user data (RCVDATA):

This attribute specifies user data that is passed to the receive exit.

You can run a sequence of receive exits. The string of user data for a series of exits must be separated by a comma, spaces, or both. For example:

```
RCVDATA(exit1_data exit2_data)
MSGDATA(exit1_data,exit2_data)
SENDDATA(exit1_data, exit2_data)
```

In WebSphere MQ for UNIX systems, and Windows systems, the length of the string of exit names and strings of user data is limited to 500 characters. In WebSphere MQ for IBM i, you can specify up to 10 exit names and the length of user data for each is limited to 32 characters. In WebSphere MQ for z/OS, you can specify up to eight strings of user data each of length 32 characters.

This attribute is valid for all channel types.

Security exit name (SCYEXIT):

This attribute specifies the name of the exit program to be run by the channel security exit.

Leave blank if no channel security exit is in effect.

The format and maximum length of the name depend on the platform, as for "Receive exit name (RCVEXIT)" on page 77. However, you can only specify one security exit.

This attribute is valid for all channel types.

Security exit user data (SCYDATA):

This attribute specifies user data that is passed to the security exit.

The maximum length is 32 characters.

This attribute is valid for all channel types.

Send exit name (SENDEXIT):

This attribute specifies the name of the exit program to be run by the channel send exit.

This attribute can be a list of names of programs that are to be run in sequence. Leave blank if no channel send exit is in effect.

The format and maximum length of this attribute depend on the platform, as for "Receive exit name (RCVEXIT)" on page 77.

This attribute is valid for all channel types.

Send exit user data (SENDDATA):

This attribute specifies user data that is passed to the send exit.

You can run a sequence of send exits. The limitations on the user data length and an example of how to specify SENDDATA for more than one exit, are as shown for RCVDATA. See "Receive exit user data (RCVDATA)" on page 77.

This attribute is valid for all channel types.

Sequence number wrap (SEQWRAP):

This attribute specifies the highest number the message sequence number reaches before it restarts at 1.

The value of the number must be high enough to avoid a number being reissued while it is still being used by an earlier message. The two ends of a channel must have the same sequence number wrap value when a channel starts; otherwise, an error occurs.

The value can be set from 100 through 999 999 999.

This attribute is valid for channel types of:
- Sender
- Server
- Receiver
- Requester
- Cluster sender
- Cluster receiver

Short retry count (SHORTRTY):

This attribute specifies the maximum number of times that the channel is to try allocating a session to its partner.

If the initial allocation attempt fails, the *short retry count* is decremented and the channel retries the remaining number of times with an interval, defined in the *short retry interval* attribute, between each attempt. If it still fails, it retries *long retry count* number of times with an interval of *long retry interval* between each attempt. If it is still unsuccessful, the channel terminates.

(Retry is not attempted if the cause of failure is such that a retry is not likely to be successful.)

If the channel initiator (on z/OS) or the channel (on distributed platforms) is stopped while the channel is retrying, the *short retry count* and *long retry count* are reset when the channel initiator or the channel is restarted, or when a message is successfully put at the sender channel. However, if the channel initiator (on z/OS) or queue manager (on distributed platforms) is shut down and restarted, the *short retry count* and *long retry count* are not reset. The channel retains the retry count values it had before the queue manager restart or the message being put.

Note: For UNIX systems, and Windows systems:
1. When a channel goes from RETRYING state to RUNNING state, the *short retry count* and *long retry count* are not reset immediately. They are reset only once the first message flows across the channel successfully after the channel went into RUNNING state, that is; once the local channel confirms the number of messages sent to the other end.
2. The *short retry count* and *long retry count* are reset when the channel is restarted.

This attribute can be set from zero through 999 999 999.

This attribute is valid for channel types of:
- Sender
- Server
- Cluster sender
- Cluster receiver

Note: On UNIX systems, and Windows systems, in order for retry to be attempted a channel initiator must be running. The channel initiator must be monitoring the initiation queue specified in the definition of the transmission queue that the channel is using.

Short retry interval (SHORTTMR):

This attribute specifies the approximate interval in seconds that the channel is to wait before retrying to establish connection, during the short retry mode.

The interval between retries might be extended if the channel has to wait to become active.

This attribute can be set from zero through 999 999.

This attribute is valid for channel types of:

- Sender
- Server
- Cluster sender
- Cluster receiver

SSL Cipher Specification (SSLCIPH):

This attribute specifies a single CipherSpec for a TLS or SSL connection.

Every WebSphere MQ channel definition includes the SSLCIPH attribute. The value is a string with a maximum length of 32 characters.

Note the following:

- The SSLCIPH attribute can contain a blank value, meaning that you are not using SSL or TLS. If one end of the channel has a blank SSLCIPH attribute, the other end of the channel must also have a blank SSLCIPH attribute.
- Alternatively, if SSLCIPH contains a nonblank value, the channel attempts to use the specified cipher to utilize SSL or TLS. Again, in this case, both ends of the channel must specify the same SSLCIPH value.

It is valid only for channels with a transport type (TRPTYPE) of TCP. If the TRPTYPE is not TCP, the data is ignored and no error message is issued.

For more information about SSLCIPH, see DEFINE CHANNEL and Specifying CipherSpecs.

SSL Client Authentication (SSLCAUTH):

This attribute specifies whether the channel needs to receive and authenticate an SSL certificate from an SSL client.

Possible values are:

OPTIONAL
If the peer SSL client sends a certificate, the certificate is processed as normal but authentication does not fail if no certificate is sent.

REQUIRED
If the SSL client does not send a certificate, authentication fails.

The default value is REQUIRED.

You can specify a value for SSLCAUTH on a non-SSL channel definition, one on which SSLCIPH is missing or blank. You can use this to temporarily disable SSL for debugging without first having to clear and then reinput the SSL parameters.

SSLCAUTH is an optional attribute.

This attribute is valid on all channel types that can ever receive a channel initiation flow, except for sender channels.

This attribute is valid for channel types of:

- Server
- Receiver
- Requester
- Server connection
- Cluster receiver

For more information about SSLCAUTH, see MQSC reference and Security.

SSL Peer (SSLPEER):

This attribute is used to check the Distinguished Name (DN) of the certificate from the peer queue manager or client at the other end of a IBM WebSphere MQ channel.

Note: An alternative way of restricting connections into channels by matching against the SSL or TLS Subject Distinguished Name, is to use channel authentication records. With channel authentication records, different SSL or TLS Subject Distinguished Name patterns can be applied to the same channel. If both SSLPEER on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns in order to connect.

If the DN received from the peer does not match the SSLPEER value, the channel does not start.

SSLPEER is an optional attribute. If a value is not specified, the peer DN is not checked when the channel is started.

On z/OS, the maximum length of the attribute is 256 bytes. On all other platforms, it is 1024 bytes.

On z/OS, the attribute values used are not checked. If you enter incorrect values, the channel fails at startup, and error messages are written to the error log at both ends of the channel. A Channel SSL Error event is also generated at both ends of the channel. On platforms that support SSLPEER, other than z/OS, the validity of the string is checked when it is first entered.

You can specify a value for SSLPEER on a non-SSL channel definition, one on which SSLCIPH is missing or blank. You can use this to temporarily disable SSL for debugging without having to clear and later reinput the SSL parameters.

For more information about using SSLPEER, see MQSC reference and Security.

This attribute is valid for all channel types.

Related information:
Channel authentication records

Transmission queue name (XMITQ):

This attribute specifies the name of the transmission queue from which messages are retrieved.

This attribute is required for channels of type sender or server, it is not valid for other channel types.

Provide the name of the transmission queue to be associated with this sender or server channel, that corresponds to the queue manager at the far side of the channel. You can give the transmission queue the same name as the queue manager at the remote end.

This attribute is valid for channel types of:

- Sender
- Server

Transport type (TRPTYPE):

This attribute specifies the transport type to be used.

The possible values are:

LU62	LU 6.2
TCP	TCP/IP
NETBIOS	NetBIOS (1)
SPX	SPX (1)
Notes: 1. For use on Windows . Can also be used on z/OS for defining client-connection channels for use on Windows.	

This attribute is valid for all channel types.

Use Dead-Letter Queue (USEDLQ):

This attribute determines whether the dead-letter queue (or undelivered message queue) is used when messages cannot be delivered by channels.

Possible values are:

NO Messages that cannot be delivered by a channel are treated as a failure. The channel either discards these messages, or the channel ends, in accordance with the setting of NPMSPEED.

YES (default)
If the queue manager DEADQ attribute provides the name of a dead-letter queue, then it is used, otherwise the behaviour is as for NO.

User ID (USERID):

This attribute specifies the user ID to be used by the MCA when attempting to initiate a secure SNA session with a remote MCA.

You can specify a task user identifier of 20 characters.

It is valid for channel types of sender, server, requester, or client-connection.

This attribute does not apply to WebSphere MQ for z/OS except for client-connection channels.

On the receiving end, if passwords are kept in encrypted format and the LU 6.2 software is using a different encryption method, an attempt to start the channel fails with invalid security details. You can avoid this failure by modifying the receiving SNA configuration to either:

- Turn off password substitution, or
- Define a security user ID and password.

On WebSphere MQ for z/OS, this attribute is valid only for client connection channels. On other platforms, it is valid for channel types of:

- Sender
- Server
- Requester
- Client connection
- Cluster sender

WebSphere MQ cluster commands

The WebSphere MQ Script commands **runmqsc** commands have special attributes and parameters that apply to clusters. There are other administrative interfaces you can use to manager clusters.

The MQSC commands are shown as they would be entered by the system administrator at the command console. Remember that you do not have to issue the commands in this way. There are a number of other methods, depending on your platform; for example:

- On WebSphere MQ for IBM i, you run MQSC commands interactively from option 26 of **WRKMQM**. You can also use CL commands or you can store MQSC commands in a file and use the **STRMQMMQSC** CL command.
- On z/OS you can use the `COMMAND` function of the **CSQUTIL** utility, the operations and control panels or you can use the z/OS console.
- On all other platforms, you can store the commands in a file and use **runmqsc**.

In a MQSC command, a cluster name, specified using the `CLUSTER` attribute, can be up to 48 characters long.

A list of cluster names, specified using the `CLUSNL` attribute, can contain up to 256 names. To create a cluster namelist, use the `DEFINE NAMELIST` command.

WebSphere MQ Explorer

The Explorer GUI can administer a cluster with repository queue managers on WebSphere MQ for z/OS Version 6 or later. You do not need to nominate an additional repository on a separate system. For earlier versions of WebSphere MQ for z/OS, the WebSphere MQ Explorer cannot administer a cluster with repository queue managers. You must therefore nominate an additional repository on a system that the WebSphere MQ Explorer can administer.

On WebSphere MQ for Windows and WebSphere MQ for Linux, you can also use WebSphere MQ Explorer to work with clusters. You can also use the stand-alone WebSphere MQ Explorer client.

Using the WebSphere MQ Explorer you can view cluster queues and inquire about the status of cluster-sender and cluster-receiver channels. WebSphere MQ Explorer includes two wizards, which you can use to guide you through the following tasks:

- Create a cluster
- Join an independent queue manager to a cluster

Programmable command formats (PCF)

Table 19. PCF equivalents of MQSC commands specifically to work with clusters

runmqsc command	**PCF equivalent**
DISPLAY CLUSQMGR	MQCMD_INQUIRE_CLUSTER_Q_MGR
SUSPEND QMGR	MQCMD_SUSPEND_Q_MGR_CLUSTER
RESUME QMGR	MQCMD_RESUME_Q_MGR_CLUSTER
REFRESH CLUSTER	MQCMD_REFRESH_CLUSTER
RESET CLUSTER	MQCMD_RESET_CLUSTER

Related concepts:

"WebSphere MQ cluster commands" on page 83
The WebSphere MQ Script commands **runmqsc** commands have special attributes and parameters that apply to clusters. There are other administrative interfaces you can use to manager clusters.

Queue-manager definition commands

Cluster attributes that can be specified on queue manager definition commands.

To specify that a queue manager holds a full repository for a cluster, use the ALTER QMGR command specifying the attribute REPOS(*clustername*). To specify a list of several cluster names, define a cluster namelist and then use the attribute REPOSNL(*namelist*) on the ALTER QMGR command:

```
DEFINE NAMELIST(CLUSTERLIST)
       DESCR('List of clusters whose repositories I host')
       NAMES(CLUS1, CLUS2, CLUS3)
 ALTER QMGR REPOSNL(CLUSTERLIST)
```

You can provide additional cluster attributes on the ALTER QMGR command

CLWLEXIT(*name***)**
Specifies the name of a user exit to be called when a message is put to a cluster queue.

CLWLDATA(*data***)**
Specifies the data to be passed to the cluster workload user exit.

CLWLLEN(*length***)**
Specifies the maximum amount of message data to be passed to the cluster workload user exit

CLWLMRUC(*channels***)**
Specifies the maximum number of outbound cluster channels.

CLWLMRUC is a local queue manager attribute that is not propagated around the cluster. It is made available to cluster workload exits and the cluster workload algorithm that chooses the destination for messages.

CLWLUSEQ(LOCAL|ANY)
Specifies the behavior of MQPUT when the target queue has both a local instance and at least one remote cluster instance. If the put originates from a cluster channel, this attribute does not apply. It is possible to specify CLWLUSEQ as both a queue attribute and a queue manager attribute.

If you specify ANY, both the local queue and the remote queues are possible targets of the MQPUT.

If you specify LOCAL, the local queue is the only target of the MQPUT.

The equivalent PCFs are MQCMD_CHANGE_Q_MGR and MQCMD_INQUIRE_Q_MGR.

Channel definition commands

Cluster attributes that can be specified on channel definition commands.

The DEFINE CHANNEL, ALTER CHANNEL, and DISPLAY CHANNEL commands have two specific CHLTYPE parameters for clusters: CLUSRCVR and CLUSSDR. To define a cluster-receiver channel you use the DEFINE CHANNEL command, specifying CHLTYPE(CLUSRCVR). Many attributes on a cluster-receiver channel definition are the same as the attributes on a receiver or sender-channel definition. To define a cluster-sender channel you use the DEFINE CHANNEL command, specifying CHLTYPE(CLUSSDR), and many of the same attributes as you use to define a sender-channel.

It is no longer necessary to specify the name of the full repository queue manager when you define a cluster-sender channel. If you know the naming convention used for channels in your cluster, you can make a CLUSSDR definition using the +QMNAME+ construction. The +QMNAME+ construction is not supported on z/OS. After connection, WebSphere MQ changes the name of the channel and substitutes the correct full repository queue manager name in place of +QMNAME+. The resulting channel name is truncated to 20 characters.

For more information on naming conventions, see Cluster naming conventions.

The technique works only if your convention for naming channels includes the name of the queue manager. For example, you define a full repository queue manager called QM1 in a cluster called CLUSTER1 with a cluster-receiver channel called CLUSTER1.QM1.ALPHA. Every other queue manager can define a cluster-sender channel to this queue manager using the channel name, CLUSTER1.+QMNAME+.ALPHA.

If you use the same naming convention for all your channels, be aware that only one +QMNAME+ definition can exist at one time.

The following attributes on the DEFINE CHANNEL and ALTER CHANNEL commands are specific to cluster channels:

CLUSTER
The CLUSTER attribute specifies the name of the cluster with which this channel is associated. Alternatively use the CLUSNL attribute.

CLUSNL The CLUSNL attribute specifies a namelist of cluster names.

NETPRTY
Cluster-receivers only.

The NETPRTY attribute specifies a network priority for the channel. NETPRTY helps the workload management routines. If there is more than one possible route to a destination, the workload management routine selects the one with the highest priority.

CLWLPRTY
The CLWLPRTY parameter applies a priority factor to channels to the same destination for workload management purposes. This parameter specifies the priority of the channel for the purposes of cluster workload distribution. The value must be in the range zero through 9, where zero is the lowest priority and 9 is the highest.

CLWLRANK
The CLWLRANK parameter applies a ranking factor to a channel for workload management

purposes. This parameter specifies the rank of a channel for the purposes of cluster workload distribution. The value must be in the range zero through 9, where zero is the lowest rank and 9 is the highest.

CLWLWGHT
The CLWLWGHT parameter applies a weighting factor to a channel for workload management purposes.CLWLWGHT weights the channel so that the proportion of messages sent down that channel can be controlled. The cluster workload algorithm uses CLWLWGHT to bias the destination choice so that more messages can be sent over a particular channel. By default all channel weight attributes are the same default value. The weight attribute allows you to allocate a channel on a powerful UNIX machine a larger weight than another channel on small desktop PC. The greater weight means that the cluster workload algorithm selects the UNIX machine more frequently than the PC as the destination for messages.

CONNAME
The CONNAME specified on a cluster-receiver channel definition is used throughout the cluster to identify the network address of the queue manager. Take care to select a value for the CONNAME parameter that resolves throughout your WebSphere MQ cluster. Do not use a generic name. Remember that the value specified on the cluster-receiver channel takes precedence over any value specified in a corresponding cluster-sender channel.

These attributes on the DEFINE CHANNEL command and ALTER CHANNEL command also apply to the DISPLAY CHANNEL command.

Note: Auto-defined cluster-sender channels take their attributes from the corresponding cluster-receiver channel definition on the receiving queue manager. Even if there is a manually defined cluster-sender channel, its attributes are automatically modified to ensure that they match the attributes on the corresponding cluster-receiver definition. Beware that you can, for example, define a CLUSRCVR without specifying a port number in the CONNAME parameter, while manually defining a CLUSSDR that does specify a port number. When the auto-defined CLUSSDR replaces the manually defined one, the port number (taken from the CLUSRCVR) becomes blank. The default port number would be used and the channel would fail.

Note: The DISPLAY CHANNEL command does not display auto-defined channels. However, you can use the DISPLAY CLUSQMGR command to examine the attributes of auto-defined cluster-sender channels.

Use the DISPLAY CHSTATUS command to display the status of a cluster-sender or cluster-receiver channel. This command gives the status of both manually defined channels and auto-defined channels.

The equivalent PCFs are MQCMD_CHANGE_CHANNEL, MQCMD_COPY_CHANNEL, MQCMD_CREATE_CHANNEL, and MQCMD_INQUIRE_CHANNEL.

Omitting the CONNAME value on a CLUSRCVR definition

In some circumstances you can omit the CONNAME value on a CLUSRCVR definition. You must not omit the CONNAME value on z/OS.

On AIX, HP-UX, IBM i, Linux, Solaris, and Windows platforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, WebSphere MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:
(1415)

The generated CONNAME is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

This facility is useful when you have machines using Dynamic Host Configuration Protocol (DHCP). If you do not supply a value for the CONNAME on a CLUSRCVR channel, you do not need to change the CLUSRCVR definition. DHCP allocates you a new IP address.

If you specify a blank for the CONNAME on the CLUSRCVR definition, WebSphere MQ generates a CONNAME from the IP address of the system. Only the generated CONNAME is stored in the repositories. Other queue managers in the cluster do not know that the CONNAME was originally blank.

If you issue the DISPLAY CLUSQMGR command you see the generated CONNAME. However, if you issue the DISPLAY CHANNEL command from the local queue manager, you see that the CONNAME is blank.

If the queue manager is stopped and restarted with a different IP address, because of DHCP, WebSphere MQ regenerates the CONNAME and updates the repositories accordingly.

Queue definition commands

Cluster attributes that can be specified on the queue definition commands.

The cluster attributes on the DEFINE QLOCAL, DEFINE QREMOTE, and DEFINE QALIAS commands, and the three equivalent ALTER commands, are:

CLUSTER
Specifies the name of the cluster to which the queue belongs.

CLUSNL Specifies a namelist of cluster names.

DEFBIND
Specifies the binding to be used when an application specifies MQOO_BIND_AS_Q_DEF on the MQOPEN call. The options for this attribute are:

- Specify DEFBIND(OPEN) to bind the queue handle to a specific instance of the cluster queue when the queue is opened. DEFBIND(OPEN) is the default for this attribute.
- Specify DEFBIND(NOTFIXED) so that the queue handle is not bound to any instance of the cluster queue.
- Specify DEFBIND(GROUP) to allow an application to request that a group of messages are all allocated to the same destination instance.

When multiple queues with the same name are advertised in a Queue Manager Cluster, applications can choose whether to send all messages from this application to a single instance (MQOO_BIND_ON_OPEN), to allow the workload management algorithm to select the most suitable destination on a per message basis (MQOO_BIND_NOT_FIXED), or allow an application to request that a 'group' of messages be all allocated to the same destination instance (MQOO_BIND_ON_GROUP). The workload balancing is re-driven between groups of messages (without requiring an MQCLOSE and MQOPEN of the queue).

When you specify DEFBIND on a queue definition, the queue is defined with one of the attributes, MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED, or MQBND_BIND_ON_GROUP. Either MQBND_BIND_ON_OPEN or MQBND_BIND_ON_GROUP must be specified when using groups with clusters.

We recommend that you set the DEFBIND attribute to the same value on all instances of the same cluster queue. Because MQOO_BIND_ON_GROUP is new in IBM WebSphere MQ Version 7.1, it must not be used if any of the applications opening this queue are connecting to IBM WebSphere MQ Version 7.0.1 or earlier queue managers.

CLWLRANK
Applies a ranking factor to a queue for workload management purposes. CLWLRANK parameter is not supported on model queues. The cluster workload algorithm selects a destination queue with the highest rank. By default CLWLRANK for all queues is set to zero.

If the final destination is a queue manager on a different cluster, you can set the rank of any intermediate gateway queue managers at the intersection of neighboring clusters. With the

intermediate queue managers ranked, the cluster workload algorithm correctly selects a destination queue manager nearer the final destination.

The same logic applies to alias queues. The rank selection is made before the channel status is checked, and therefore even non-accessible queue managers are available for selection. This has the effect of allowing a message to be routed through a network, rather than having it select between two possible destinations (as the priority would). So, if a channel is not started to the place where the rank has indicated, the message is not routed to the next highest rank, but waits until a channel is available to that destination (the message is held on the transmit queue).

CLWLPRTY

Applies a priority factor to a queue for workload management purposes. The cluster workload algorithm selects a destination queue with the highest priority. By default priority for all queues is set to zero.

If there are two possible destination queues, you can use this attribute to make one destination failover to the other destination. The priority selection is made after the channel status is checked. All messages are sent to the queue with the highest priority unless the status of the channel to that destination is not as favorable as the status of channels to other destinations. This means that only the most accessible destinations are available for selection. This has the effect of prioritizing between multiple destinations that are all available.

CLWLUSEQ

Specifies the behavior of an MQPUT operation for a queue. This parameter specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance (except where the MQPUT originates from a cluster channel). This parameter is only valid for local queues.

Possible values are: `QMGR` (the behavior is as specified by the `CLWLUSEQ` parameter of the queue manager definition), ANY (the queue manager treats the local queue as another instance of the cluster queue, for the purposes of workload distribution), `LOCAL` (the local queue is the only target of the MQPUT operation, providing the local queue is put enabled). The MQPUT behavior depends upon the cluster workload management algorithm.

The attributes on the `DEFINE QLOCAL`, `DEFINE QREMOTE`, and `DEFINE QALIAS` commands also apply to the `DISPLAY QUEUE` command.

To display information about cluster queues, specify a queue type of `QCLUSTER` or the keyword `CLUSINFO` on the `DISPLAY QUEUE` command, or use the command `DISPLAY QCLUSTER`.

The `DISPLAY QUEUE` or `DISPLAY QCLUSTER` command return the name of the queue manager that hosts the queue (or the names of all queue managers if there is more than one instance of the queue). It also returns the system name for each queue manager that hosts the queue, the queue type represented, and the date and time at which the definition became available to the local queue manager. This information is returned using the `CLUSQMGR`, `QMID`, `CLUSQT`, `CLUSDATE`, and `CLUSTIME` attributes.

The system name for the queue manager (`QMID`), is a unique, system-generated name for the queue manager.

You can define a cluster queue that is also a shared queue. For example. on z/OS you can define:

```
DEFINE QLOCAL(MYQUEUE) CLUSTER(MYCLUSTER) QSGDISP(SHARED) CFSTRUCT(STRUCTURE)
```

The equivalent PCFs are `MQCMD_CHANGE_Q`, `MQCMD_COPY_Q`, `MQCMD_CREATE_Q`, and `MQCMD_INQUIRE_Q`.

DISPLAY CLUSQMGR

Use the DISPLAY CLUSQMGR command to display cluster information about queue managers in a cluster.

If you issue this command from a queue manager with a full repository, the information returned applies to every queue manager in the cluster. Otherwise the information returned applies only to the queue managers in which it has an interest. That is, every queue manager to which it has tried to send a message and every queue manager that holds a full repository.

The information includes most channel attributes that apply to cluster-sender and cluster-receiver channels. In addition, the following attributes can be displayed:

DEFTYPE

How the queue manager was defined. DEFTYPE can be one of the following values:

CLUSSDR

A cluster sender-channel has been administratively defined on the local queue manager but not yet recognized by the target queue manager. To be in this state the local queue manager has defined a manual cluster-sender channel but the receiving queue manager has not accepted the cluster information. This may be due to the channel never having been established due to availability or to an error in the cluster-sender configuration, for example a mismatch in the CLUSTER property between the sender and receiver definitions. This is a transitory condition or error state and should be investigated.

CLUSSDRA

This value represents an automatically discovered cluster queue manager, no cluster-sender channel is defined locally. This is the DEFTYPE for cluster queue managers for which the local queue manager has no local configuration but has been informed of. For example

- If the local queue manager is a full repository queue manager it should be the DEFTYPE value for all partial repository queue managers in the cluster.
- If the local queue manager is a partial repository, this could be the host of a cluster queue that is being used from this local queue manager or from a second full repository queue manager that this queue manager has been told to work with.

If the DEFTYPE value is CLUSSDRA and the local and remote queue managers are both full repositories for the named cluster, the configuration is not correct as a locally defined cluster-sender channel must be defined to convert this to a DEFTYPE of CLUSSDRB.

CLUSSDRB

A cluster sender-channel has been administratively defined on the local queue manager and accepted as a valid cluster channel by the target queue manager. This is the expected DEFTYPE of a partial repository queue manager's manually configured full repository queue manager. It should also be the DEFTYPE of any CLUSQMGR from one full repository to another full repository in the cluster. Manual cluster-sender channels should not be configured to partial repositories or from a partial repository queue manager to more than one full repository. If a DEFTYPE of CLUSSDRB is seen in either of these situations it should be investigated and corrected.

CLUSRCVR

Administratively defined as a cluster-receiver channel on the local queue manager. This represents the local queue manager in the cluster.

Note: To identify which CLUSQMGRs are full repository queue managers for the cluster, see the QMTYPE property.

For more information on defining cluster channels, see Cluster channels.

QMTYPE Whether it holds a full repository or only a partial repository.

CLUSDATE
The date at which the definition became available to the local queue manager.

CLUSTIME
The time at which the definition became available to the local queue manager.

STATUS The status of the cluster-sender channel for this queue manager.

SUSPEND
Whether the queue manager is suspended.

CLUSTER
What clusters the queue manager is in.

CHANNEL
The cluster-receiver channel name for the queue manager.

XMITQ The cluster transmission queue used by the queue manager. The property is only available on platforms other than z/OS.

SUSPEND QMGR, RESUME QMGR and clusters

Use the SUSPEND QMGR and RESUME QMGR command to temporarily reduce the inbound cluster activity to this queue manager, for example, before you perform maintenance on this queue manager, and then reinstate it.

While a queue manager is suspended from a cluster, it does not receive messages on cluster queues that it hosts if there is an available queue of the same name on an alternative queue manager in the cluster. However, messages that are explicitly targeted at this queue manager, or where the target queue is only available on this queue manager, are still directed to this queue manager.

Receiving further inbound messages while the queue manager is suspended can be prevented by stopping the cluster receiver channels for this cluster. To stop the cluster receiver channels for a cluster, use the FORCE mode of the SUSPEND QMGR command.

Related information:

SUSPEND QMGR
Use the MQSC command SUSPEND QMGR to advise other queue managers in a cluster to avoid sending messages to the local queue manager if possible, or to suspend logging and update activity for the queue manager until a subsequent RESUME QMGR command is issued. Its action can be reversed by the RESUME QMGR command. This command does not mean that the queue manager is disabled.

RESUME QMGR
Use the MQSC command RESUME QMGR to inform other queue managers in a cluster that the local queue manager is available again for processing and can be sent messages. It reverses the action of the SUSPEND QMGR command.

Maintaining a queue manager

REFRESH CLUSTER

Issue the REFRESH CLUSTER command from a queue manager to discard all locally held information about a cluster. You are unlikely to need to use this command, except in exceptional circumstances.

There are three forms of this command:

REFRESH CLUSTER(clustername) REPOS(NO)
The default. The queue manager retains knowledge of all locally defined cluster queue manager and cluster queues and all cluster queue managers that are full repositories. In addition, if the queue manager is a full repository for the cluster it also retains knowledge of the other cluster queue managers in the cluster. Everything else is removed from the local copy of the repository and rebuilt from the other full repositories in the cluster. Cluster channels are not stopped if REPOS(NO) is used. A full repository uses its CLUSSDR channels to inform the rest of the cluster that it has completed its refresh.

REFRESH CLUSTER(clustername) REPOS(YES)

In addition to the default behavior, objects representing full repository cluster queue managers are also refreshed. It is not valid to use this option if the queue manager is a full repository, if used the command will fail with an error AMQ9406/CSQX406E logged. If it is a full repository, you must first alter it so that it is not a full repository for the cluster in question. The full repository location is recovered from the manually defined CLUSSDR definitions. After refreshing with REPOS(YES) has been issued the queue manager can be altered so that it is once again a full repository, if required.

REFRESH CLUSTER(*)

Refreshes the queue manager in all the clusters it is a member of. If used with REPOS(YES) REFRESH CLUSTER(*) has the additional effect of forcing the queue manager to restart its search for full repositories from the information in the local CLUSSDR definitions. The search takes place even if the CLUSSDR channel connects the queue manager to several clusters.

Note: Use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, for example by creating a sudden increase in work for the full repositories as they process the repropagation of a queue manager cluster resources. For such reasons it is best to avoid use of the command in day-to-day work if possible and use alternative methods to correct specific inconsistencies.

Related information:

Clustering: REFRESH CLUSTER and the history queue

RESET CLUSTER

Use the RESET CLUSTER command to forcibly remove a queue manager from a cluster in exceptional circumstances.

You are unlikely to need to use this command, except in exceptional circumstances.

You can issue the RESET CLUSTER command only from full repository queue managers. The command takes two forms, depending on whether you reference the queue manager by name or identifier.

1. RESET CLUSTER(*clustername*
) QMNAME(*qmname*) ACTION(FORCEREMOVE) QUEUES(NO)
2. RESET CLUSTER(*clustername*
) QMID(*qmid*) ACTION(FORCEREMOVE) QUEUES(NO)

You cannot specify both QMNAME and QMID . If you use QMNAME, and there is more than one queue manager in the cluster with that name, the command is not run. Use QMID instead of QMNAME to ensure the RESET CLUSTER command is run.

Specifying QUEUES(NO) on a RESET CLUSTER command is the default. Specifying QUEUES(YES) removes references to cluster queues owned by the queue manager from the cluster. The references are removed in addition to removing the queue manager from the cluster itself.

The references are removed even if the cluster queue manager is not visible in the cluster; perhaps because it was previously forcibly removed, without the QUEUES option.

You might use the RESET CLUSTER command if, for example, a queue manager has been deleted but still has cluster-receiver channels defined to the cluster. Instead of waiting for WebSphere MQ to remove these definitions (which it does automatically) you can issue the RESET CLUSTER command to tidy up sooner. All other queue managers in the cluster are then informed that the queue manager is no longer available.

If a queue manager is temporarily damaged, you might want to tell the other queue managers in the cluster before they try to send it messages. **RESET CLUSTER** removes the damaged queue manager. Later, when the damaged queue manager is working again, use the **REFRESH CLUSTER** command to reverse the effect of **RESET CLUSTER** and return the queue manager to the cluster. If the queue manager is in a

publish/subscribe cluster, you then need to issue the REFRESH QMGR TYPE(PROXYSUB) command to reinstate any required proxy subscriptions. See REFRESH CLUSTER considerations for publish/subscribe clusters.

Using the RESET CLUSTER command is the only way to delete auto-defined cluster-sender channels. You are unlikely to need this command in normal circumstances. The IBM Support Center might advise you to issue the command to tidy up the cluster information held by cluster queue managers. Do not use this command as a short cut to removing a queue manager from a cluster. The correct way to remove a queue manager from a cluster is described in Removing a queue manager from a cluster.

Because repositories retain information for only 90 days, after that time a queue manager that was forcibly removed can reconnect to a cluster. It reconnects automatically, unless it has been deleted. If you want to prevent a queue manager from rejoining a cluster, you need to take appropriate security measures.

All cluster commands, except DISPLAY CLUSQMGR, work asynchronously. Commands that change object attributes involving clustering update the object and send a request to the repository processor. Commands for working with clusters are checked for syntax, and a request is sent to the repository processor.

The requests sent to the repository processor are processed asynchronously, along with cluster requests received from other members of the cluster. Processing might take a considerable time if they have to be propagated around the whole cluster to determine if they are successful or not.

Workload balancing

If a cluster contains more than one instance of the same queue, WebSphere MQ selects a queue manager to route a message to. It uses the cluster workload management algorithm to determine the best queue manager to use. You can provide the workload balancing algorithm to select the queue manager by writing a cluster workload exit program.

Suitable destinations are chosen based on the availability of the queue manager and queue, and on a number of cluster workload-specific attributes associated with queue managers, queues and channels.

If the results of the workload balancing algorithm do not meet your needs, you can write a cluster workload user exit program. Use the exit to route messages to the queue of your choice in the cluster.

The cluster workload management algorithm:

The workload management algorithm uses workload balancing attributes and many rules to select the final destination for messages being put onto cluster queues.

This section lists the workload management algorithm used when determining the final destination for messages being put onto cluster queues. These rules are influenced by the settings applied to the following attributes for queues, queue managers, and channels:

Table 20. Attributes for cluster workload management

Queues	Queue managers	Channels
• CLWLPRTY • CLWLRANK • CLWLUSEQ	• CLWLUSEQ • CLWLMRUC	• CLWLPRTY • CLWLRANK • CLWLWGHT • NETPRTY

Initially, the queue manager builds a list of possible destinations from two procedures:

- Matching the target ObjectName and ObjectQmgrName with queue manager alias definitions that are shared in the same clusters as the queue manager.
- Finding unique routes, or in other words, channels, to a queue manager that hosts a queue with the name ObjectName and is in one of the clusters that the queue manager is a member of.

The steps of the algorithm that follow eliminate destinations from the list of possible destinations.

1. If a queue name is specified:
   a. Queues that are not put enabled are eliminated as possible destinations.
   b. Remote instances of queues that do not share a cluster with the local queue manager are eliminated.
   c. Remote CLUSRCVR channels that are not in the same cluster as the queue are eliminated.
2. If a queue manager name is specified,
   a. Queue manager aliases that are not put enabled are eliminated.
   b. Remote CLUSRCVR channels that are not in the same cluster as the local queue manager are eliminated.
3. If the resulting set of queues contains the local instance of the queue, the local instance of a queue is typically used. The local instance of the queue is used if one of these three conditions are true:
   - The use-queue attribute of the queue, CLWLUSEQ is set to LOCAL.
   - Both the following are true:
     a. The use-queue attribute of the queue, CLWLUSEQ is set to QMGR.
     b. The use-queue attribute of the queue manager, CLWLUSEQ is set to LOCAL.
   - The message is received over a cluster channel rather than by being put by a local application.

     Note: You can detect a message from a cluster channel in a user exit if the both the MQWXP_PUT_BY_CLUSTER_CH and MQQF_CLWL_USEQ_ANY flags are not set:
     – MQWXP.Flags flag MQWXP_PUT_BY_CLUSTER_CH.
     – MQWQR.QFlags flag MQQF_CLWL_USEQ_ANY.
4. If the message is a cluster PCF message, any queue manager to which a publication or subscription has already been sent is eliminated.
5. All channels to queue managers or queue manager alias with a CLWLRANK less than the maximum rank of all remaining channels or queue manager aliases are eliminated.
6. All queues (not queue manager aliases) with a CLWLRANK less than the maximum rank of all remaining queues are eliminated.
7. If only remote instances of a queue remain, resumed queue managers are chosen in preference to suspended ones.
8. If more than one remote instance of a queue remains, all channels that are inactive or running are included. The state constants are listed:
   - MQCHS_INACTIVE
   - MQCHS_RUNNING
9. If no remote instance of a queue remains, all channels that are in binding, initializing, starting, or stopping state are included. The state constants are listed:
   - MQCHS_BINDING
   - MQCHS_INITIALIZING
   - MQCHS_STARTING
   - MQCHS_STOPPING
10. If no remote instance of a queue remains, all channels that are being tried again, MQCHS_RETRYING are included.
11. If no remote instance of a queue remains, all channels in requesting, paused, or stopped state are included. The state constants are listed:
    - MQCHS_REQUESTING

- MQCHS_PAUSED
- MQCHS_STOPPED

12. If more than one remote instance of a queue remains and the message is a cluster PCF message, locally defined CLUSSDR channels are chosen.
13. If more than one remote instance of a queue to any queue manager remains, channels with the highest NETPRTY value for each queue manager are chosen.
14. If a queue manager is being chosen:
    - All remaining channels and queue manager aliases other than channels and aliases with the highest priority, CLWLPRTY, are eliminated. If any queue manager aliases remain, channels to the queue manager are kept.
15. If a queue is being chosen:
    - All queues other than queues with the highest priority, CLWLPRTY, are eliminated, and channels are kept.
16. All channels, except a number of channels with the highest values in MQWDR.DestSeqNumber are eliminated. The elimination stops when the number of remaining channels is no greater than the maximum allowed number of most recently used channels, CLWLMRUC.
17. If more than one remote instance of a queue remains, the least recently used channel is chosen. The least recently used channel has the lowest value of MQWDR.DestSeqFactor.
    - If there is more than one channel with the lowest value, one of the channels with the lowest value in MQWDR.DestSeqNumber is chosen.
    - The destination sequence factor of the choice is increased by the queue manager, by approximately 1000/CLWLWGHT.

    Note:

    a. The destination sequence factors of all destinations are reset to zero if the cluster workload attributes of available CLUSRCVR channels are altered. The sequence factors are zeroed if new CLUSRCVR channels become available.
    b. Modifications to workload attributes of manually defined CLUSSDR channels do not reset the Destination Sequence Factor.

The distribution of user messages is not always exact, because administration and maintenance of the cluster causes messages to flow across channels. The result is an uneven distribution of user messages which can take some time to stabilize. Because of the admixture of administration and user messages, place no reliance on the exact distribution of messages during workload balancing.

CLWLPRTY queue attribute:

The CLWLPRTY queue attribute specifies the priority of local, remote, or alias queues for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest priority and 9 is the highest.

Use the CLWLPRTY queue attribute to set a preference for destination queues. WebSphere MQ selects the destinations with the highest priority before selecting destinations with the lowest cluster destination priority. If there are multiple destinations with the same priority, it selects the least recently used destination.

If there are two possible destinations, you can use this attribute to allow failover. The highest priority queue manager receives requests, lower priority queue managers act as reserves. If the highest priority queue manager fails, then the next highest priority queue manager that is available, takes over.

WebSphere MQ obtains the priority of queue managers after checking channel status. Only available queue managers are candidates for selection.

Note:

The availability of a remote queue manager is based on the status of the channel to that queue manager. When channels start, their state changes several times, with some of the states being less preferential to the cluster workload management algorithm. In practice this means that lower priority (backup) destinations can be chosen while the channels to higher priority (primary) destinations are starting.

If you need to ensure that no messages go to a backup destination, do not use CLWLPRTY. Consider using separate queues, or CLWLRANK with a manual switch over from the primary to backup.

CLWLRANK queue attribute:

The CLWLRANK queue attribute specifies the rank of a local, remote, or alias queue for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest rank and 9 is the highest.

Use the CLWLRANK queue attribute if you want control over the final destination for messages sent to a queue manager in another cluster. When you set CLWLRANK, messages take a specified route through the interconnected clusters towards a higher ranked destination.

For example, you might have defined two identically configured gateway queue managers to improve the availability of a gateway. Suppose you have defined cluster alias queues at the gateways for a local queue defined in the cluster. If the local queue becomes unavailable, you intend the message to be held at one of the gateways pending the queue becoming available again. To hold the queue at a gateway, you must define the local queue with a higher rank than the cluster alias queues at the gateway.

If you define the local queue with the same rank as the queue aliases and the local queue is unavailable, the message travels between the gateways. On finding the local queue unavailable the first gateway queue manager routes the message to the other gateway. The other gateway tries to deliver the message to the target local queue again. If the local queue is still unavailable, it routes the message back to the first gateway. The message keeps being moved back and forth between the gateways until the target local queue became available again. By giving the local queue a higher rank, even if the queue is unavailable, the message is not rerouted to a destination of lower rank.

WebSphere MQ obtains the rank of queues before checking channel status. Obtaining the rank before checking channel status means that even non-accessible queues are available for selection. It allows messages to be routed through the network even if the final destination is unavailable.

If you used the priority attribute WebSphere MQ selects between available destinations. If a channel is not available to the destination with the highest rank, the message is held on the transmission queue. It is released when the channel becomes available. The message does not get sent to the next available destination in the rank order.

CLWLUSEQ queue attribute:

The CLWLUSEQ queue attribute specifies whether a local instance of a queue is given preference as a destination over other instances in a cluster.

The CLWLUSEQ queue attribute is valid only for local queues. It only applies if the message is put by an application, or a channel that is not a cluster channel.

LOCAL The local queue is the only target of MQPUT, providing the local queue is put enabled. MQPUT behavior depends upon the cluster workload management.

QMGR The behavior is as specified by the CLWLUSEQ queue manager attribute.

ANY MQPUT treats the local queue the same as any other instance of the queue in the cluster for workload distribution.

CLWLUSEQ queue manager attribute:

The CLWLUSEQ queue manager attribute specifies whether a local instance of a queue is given preference as a destination over other instances of the queue in a cluster. The attribute applies if the CLWLUSEQ queue attribute is set to QMGR.

The CLWLUSEQ queue attribute is valid only for local queues. It only applies if the message is put by an application, or a channel that is not a cluster channel.

LOCAL The local queue is the only target of MQPUT. LOCAL is the default.

ANY MQPUT treats the local queue the same as any other instance of the queue in the cluster for workload distribution.

CLWLMRUC queue manager attribute:

The CLWLMRUC queue manager attribute sets the number of most recently chosen channels. The cluster workload management algorithm uses CLWLMRUC to restrict the number of active outbound cluster channels. The value must be in the range 1 - 999 999 999.

The initial default value is 999 999 999.

CLWLPRTY channel attribute:

The CLWLPRTY channel attribute specifies the priority of CLUSSDR or CLUSRCVR channels for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest priority and 9 is the highest.

Use the CLWLPRTY channel attribute to set a preference for a CLUSSDR or CLUSRCVR channel. WebSphere MQ selects the destinations with the highest priority before selecting destinations with the lowest cluster destination priority. If there are multiple destinations with the same priority, it selects the least recently used destination.

If there are two possible destinations, you can use this attribute to allow failover. Messages go to the queue manager with the highest priority channel. If it becomes unavailable then messages go to the next highest priority queue manager. Lower priority queue managers act as reserves.

WebSphere MQ obtains the priority of channels after checking channel status. Only available queue managers are candidates for selection.

Note:

The availability of a remote queue manager is based on the status of the channel to that queue manager. When channels start, their state changes several times, with some of the states being less preferential to the cluster workload management algorithm. In practice this means that lower priority (backup) destinations can be chosen while the channels to higher priority (primary) destinations are starting.

If you need to ensure that no messages go to a backup destination, do not use CLWLPRTY. Consider using separate queues, or CLWLRANK with a manual switch over from the primary to backup.

CLWLRANK channel attribute:

The CLWLRANK channel attribute specifies the rank of CLUSSDR or CLUSRCVR channels for cluster workload distribution. The value must be in the range 0-9, where 0 is the lowest rank and 9 is the highest.

Use the CLWLRANK channel attribute if you want control over the final destination for messages sent to a queue manager in another cluster. Control the choice of final destination by setting the rank of the channels connecting a queue manager to the gateway queue managers at the intersection of the clusters. When you set CLWLRANK, messages take a specified route through the interconnected clusters towards a higher ranked destination. For example, messages arrive at a gateway queue manager that can send them to either of two queue managers using channels ranked 1 and 2. They are automatically sent to the queue manager connected by a channel with the highest rank, in this case the channel to the queue manager ranked 2.

WebSphere MQ obtains the rank of channels before checking channel status. Obtaining the rank before checking channel status means that even non-accessible channels are available for selection. It allows messages to be routed through the network even if the final destination is unavailable.

If you used the priority attribute WebSphere MQ selects between available destinations. If a channel is not available to the destination with the highest rank, the message is held on the transmission queue. It is released when the channel becomes available. The message does not get sent to the next available destination in the rank order.

CLWLWGHT channel attribute:

The CLWLWGHT channel attribute specifies the weight applied to CLUSSDR and CLUSRCVR channels for cluster workload distribution. The value must be in the range 1-99, where 1 is the lowest weight and 99 is the highest.

Use CLWLWGHT to send servers with more processing power more messages. The higher the channel weight, the more messages are sent over that channel.

NETPRTY channel attribute:

The NETPRTY channel attribute specifies the priority for a CLUSRCVR channel. The value must be in the range 0-9, where 0 is the lowest priority and 9 is the highest.

Use the NETPRTY attribute to make one network the primary network, and another network the backup network. Given a set of equally ranked channels, clustering chooses the path with the highest priority when multiple paths are available.

A typical example of using the NETPRTY channel attribute is to differentiate between networks that have different costs or speeds and connect the same destinations.

Cluster workload exit call and data structures

This section provides reference information for the cluster workload exit and the data structures. This is general-use programming interface information.

You can write cluster workload exits in the following programming languages:

- C
- System/390® assembler (WebSphere MQ for z/OS)

The call is described in:

- "MQ_CLUSTER_WORKLOAD_EXIT - Call description" on page 98

The structure data types used by the exit are described in:

- "MQXCLWLN - Navigate Cluster workload records" on page 99
- "MQWXP - Cluster workload exit parameter structure" on page 102
- "MQWDR - Cluster workload destination record structure" on page 110
- "MQWQR - Cluster workload queue record structure" on page 113
- "MQWCR - Cluster workload cluster record structure" on page 118
-

Throughout this section, queue-manager attributes and queue attributes are shown in full. The equivalent names that are used in the MQSC commands book are shown below. For details of MQSC commands, see MQSC reference.

Table 21. Queue-manager attributes

Full name	Name used in MQSC
ClusterWorkloadData	CLWLDATA
ClusterWorkloadExit	CLWLEXIT
ClusterWorkloadLength	CLWLLEN

Table 22. Queue attributes

Full name	Name used in MQSC
DefBind	DEFBIND
DefPersistence	DEFPSIST
DefPriority	DEFPRTY
InhibitPut	PUT
QDesc	DESCR

MQ_CLUSTER_WORKLOAD_EXIT - Call description:

The cluster workload exit is called by the queue manager to route a message to an available queue manager.

Note: No entry point called MQ_CLUSTER_WORKLOAD_EXIT is provided by the queue manager. Instead, the name of the cluster workload exit is defined by the ClusterWorkloadExit queue-manager attribute.

The MQ_CLUSTER_WORKLOAD_EXIT exit is supported on all platforms.

Syntax

```
MQ_CLUSTER_WORKLOAD_EXIT (ExitParms)
```

Parameters for MQ_CLUSTER_WORKLOAD_EXIT:

Description of the parameters in the MQ_CLUSTER_WORKLOAD_EXIT call.

***ExitParms*(MQWXP) - input/output**

Exit parameter block.

- The exit sets information in MQWXP to indicate how to manage the workload.

Usage notes:

The function performed by the cluster workload exit is defined by the provider of the exit. The exit, however, must conform to the rules defined in the associated control block MQWXP.

No entry point called MQ_CLUSTER_WORKLOAD_EXIT is provided by the queue manager. However, a **typedef** is provided for the name MQ_CLUSTER_WORKLOAD_EXIT in the C programming language. Use the typedef to declare the user-written exit, to ensure that the parameters are correct.

Language invocations for MQ_CLUSTER_WORKLOAD_EXIT:

The MQ_CLUSTER_WORKLOAD_EXIT supports two languages, C and High Level Assembler.

C invocation

```
MQ_CLUSTER_WORKLOAD_EXIT (&ExitParms);
```

Replace *MQ_CLUSTER_WORKLOAD_EXIT* with the name of your cluster workload exit function.

Declare the *MQ_CLUSTER_WORKLOAD_EXIT* parameters as follows:

```
MQWXP  ExitParms;  /* Exit parameter block */
```

High Level Assembler invocation

```
        CALL EXITNAME,(EXITPARMS)
```

Declare the parameters as follows:

```
EXITPARMS           CMQWXPA             Exit parameter block
```

MQXCLWLN - Navigate Cluster workload records:

The MQXCLWLN call is used to navigate through the chains of MQWDR, MQWQR, and MQWCR records stored in the cluster cache.

The cluster cache is an area of main storage used to store information relating to the cluster.

If the cluster cache is static, it has a fixed size. If you set it to dynamic, the cluster cache can expand as required.

Set the type of cluster cache to STATIC or DYNAMIC using either a system parameter or macro.

- The system parameter ClusterCacheType on platforms other than z/OS
- The CLCACHE parameter in the CSQ6SYSP macro on z/OS.

Syntax

```
MQXCLWLN (ExitParms, CurrentRecord, NextOffset, NextRecord, Compcode, Reason)
```

Parameters for MQXCLWLN - Navigate Cluster workload records:

Description of the parameters in the MQXCLWLN call.

ExitParms **(MQWXP) - input/output**
Exit parameter block.

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate how to manage the workload.

CurrentRecord **(MQPTR) - input**
Address of current record.

This structure contains information relating to the address of the record currently being examined by the exit. The record must be one of the following types:

- Cluster workload destination record (MQWDR)
- Cluster workload queue record (MQWQR)
- Cluster workload cluster record (MQWCR)

NextOffset **(MQLONG) - input**
Offset of next record.

This structure contains information relating to the offset of the next record or structure. *NextOffset* is the value of the appropriate offset field in the current record, and must be one of the following fields:

- ChannelDefOffset field in MQWDR
- ClusterRecOffset field in MQWDR
- ClusterRecOffset field in MQWQR
- ClusterRecOffset field in MQWCR

NextRecord **(MQPTR) - output**
Address of next record or structure.

This structure contains information relating to the address of the next record or structure. If *CurrentRecord* is the address of an MQWDR, and *NextOffset* is the value of the ChannelDefOffset field, *NextRecord* is the address of the channel definition structure (MQCD).

If there is no next record or structure, the queue manager sets *NextRecord* to the null pointer, and the call returns completion code MQCC_WARNING and reason code MQRC_NO_RECORD_AVAILABLE.

CompCode **(MQLONG) - output**
Completion code.

The completion code has one of the following values:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason **(MQLONG) - output**
Reason code qualifying CompCode

If CompCode is MQCC_OK:

MQRC_NONE
(0, X'0000')
No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_NO_RECORD_AVAILABLE
(2359, X'0937')

No record available. An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain. The current record is the last record in the chain. Corrective action: None.

If *CompCode* is MQCC_FAILED:

MQRC_CURRENT_RECORD_ERROR
(2357, X'0935')

CurrentRecord parameter not valid. An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain. The address specified by the *CurrentRecord* parameter is not the address of a valid record.

CurrentRecord must be the address of a destination record, MQWDR, queue record (MQWQR), or cluster record (MQWCR) residing within the cluster cache. Corrective action: Ensure that the cluster workload exit passes the address of a valid record residing in the cluster cache.

MQRC_ENVIRONMENT_ERROR
(2012, X'07DC')

Call not valid in environment. An MQXCLWLN call was issued, but not from a cluster workload exit.

MQRC_NEXT_OFFSET_ERROR
(2358, X'0936')

NextOffset parameter not valid. An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain. The offset specified by the *NextOffset* parameter is not valid. *NextOffset* must be the value of one of the following fields:

- ChannelDefOffset field in MQWDR
- ClusterRecOffset field in MQWDR
- ClusterRecOffset field in MQWQR
- ClusterRecOffset field in MQWCR

Corrective action: Ensure that the value specified for the *NextOffset* parameter is the value of one of the fields listed previously.

MQRC_NEXT_RECORD_ERROR
(2361, X'0939')

NextRecord parameter not valid.

MQRC_WXP_ERROR
(2356, X'0934')

Workload exit parameter structure not valid. An MQXCLWLN call was issued from a cluster workload exit to obtain the address of the next record in the chain. The workload exit parameter structure *ExitParms* is not valid, for one of the following reasons:

- The parameter pointer is not valid. It is not always possible to detect parameter pointers that are not valid; if not detected, unpredictable results occur.
- The StrucId field is not MQWXP_STRUC_ID.
- The Version field is not MQWXP_VERSION_2.
- The Context field does not contain the value passed to the exit by the queue manager.

Corrective action: Ensure that the parameter specified for *ExitParms* is the MQWXP structure that was passed to the exit when the exit was invoked.

Usage notes for MQXCLWLN - Navigate Cluster workload records:

Use MQXCLWLN to navigate through cluster records, even if the cache is static.

If the cluster cache is dynamic, the MQXCLWLN call must be used to navigate through the records. The exit ends abnormally if simple pointer-and-offset arithmetic is used to navigate through the records.

If the cluster cache is static, MQXCLWLN need not be used to navigate through the records. Typically use MQXCLWLN even when the cache is static. You can then change the cluster cache to being dynamic without needing to change the workload exit.

Language invocations of MQXCLWLN:

MQXCLWLN supports two languages, C and High Level Assembler.

C invocation

```
MQXCLWLN (&ExitParms, CurrentRecord, NextOffset, &NextRecord, &CompCode, &Reason) ;
```

Declare the parameters as follows:

```
Typedef struct tagMQXCLWLN {
  MQWXP     ExitParms;            /* Exit parameter block */
  MQPTR     CurrentRecord;        /* Address of current record*/
  MQLONG    NextOffset;           /* Offset of next record */
  MQPTR     NextRecord;           /* Address of next record or structure */
  MQLONG    CompCode;             /* Completion code */
  MQLONG    Reason;               /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQXCLWLN,(CLWLEXITPARMS,CURRENTRECORD,NEXTOFFSET,NEXTRECORD,COMPCODE,REASON)
```

Declare the parameters as follows:

```
CLWLEXITPARMS  CMQWXPA,      Cluster workload exit parameter block
CURRENTRECORD  CMQWDRA,      Current record
NEXTOFFSET     DS  F         Next offset
NEXTRECORD     DS  F         Next record
COMPCODE       DS  F         Completion code
REASON         DS  F         Reason code qualifying COMPCODE
```

MQWXP - Cluster workload exit parameter structure:

The following table summarizes the fields in the MQWXP - Cluster workload exit parameter structure.

Table 23. Fields in MQWXP

Field	Description	Page
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
ExitId	Type of exit	ExitId
ExitReason	Reason for invoking exit	ExitReason
ExitResponse	Response from exit	ExitResponse
ExitResponse2	Secondary response from exit	ExitResponse2
Feedback	Feedback code	Feedback
Flags	Flags values. These bit flags are used to indicate information about the message being put	Flags
ExitUserArea	Exit user area	ExitUserArea

Table 23. Fields in MQWXP (continued)

Field	Description	Page
ExitData	Exit data	ExitData
MsgDescPtr	Address of message descriptor (MQMD)	MsgDescPtr
MsgBufferPtr	Address of buffer containing some or all the message data	MsgBufferPtr
MsgBufferLength	Length of buffer containing message data	MsgBufferLength
MsgLength	Length of complete message	MsgLength
QName	Name of queue	QName
QMgrName	Name of local queue manager	QMgrName
DestinationCount	Number of possible destinations	DestinationCount
DestinationChosen	Destination chosen	DestinationChosen
DestinationArrayPtr	Address of an array of pointers to destination records (MQWDR)	DestinationArrayPtr
QArrayPtr	Address of an array of pointers to queue records (MQWQR)	QArrayPtr
Note: The remaining fields are ignored if Version is less than MQWXP_VERSION_2.		
CacheContext	Context information	CacheContext
CacheType	Type of cluster cache	CacheType
Note: The remaining fields are ignored if Version is less than MQWXP_VERSION_3.		
CLWLMRUChannels	Maximum number of allowed active outbound cluster channels	CLWLMRUChannels
Note: The remaining fields are ignored if Version is less than MQWXP_VERSION_4.		
pEntryPoints	Address of the MQIEP structure to allow MQI and DCI calls to be made	pEntryPoints

The cluster workload exit parameter structure describes the information that is passed to the cluster workload exit.

The cluster workload exit parameter structure is supported on all platforms

Additionally, the MQWXP1, MQWXP2 and MQWXP3 structures are available for backwards compatibility.

Fields in MQWXP - Cluster workload exit parameter structure:

Description of the fields in the MQWXP - Cluster workload exit parameter structure

StrucId (MQCHAR4) - input
The structure identifier for the cluster workload exit parameter structure.

- The StrucId value is MQWXP_STRUC_ID.
- For the C programming language, the constant MQWXP_STRUC_ID_ARRAY is also defined. It has the same value as MQWXP_STRUC_ID. It is an array of characters instead of a string.

Version (MQLONG) - input
Indicates the structure version number. Version takes one of the following values:

MQWXP_VERSION_1
Version-1 cluster workload exit parameter structure.

MQWXP_VERSION_1 is supported in all environments.

MQWXP_VERSION_2
Version-2 cluster workload exit parameter structure.

MQWXP_VERSION_2 is supported in the following environments: AIX, HP-UX, Linux, IBM i, Solaris and Windows.

MQWXP_VERSION_3
Version-3 cluster workload exit parameter structure.

MQWXP_VERSION_3 is supported in the following environments: AIX, HP-UX, Linux, IBM i, Solaris and Windows.

MQWXP_VERSION_4
Version-4 cluster workload exit parameter structure.

MQWXP_VERSION_4 is supported in the following environments: AIX, HP-UX, Linux, IBM i, Solaris and Windows.

MQWXP_CURRENT_VERSION
Current version of cluster workload exit parameter structure.

ExitId (MQLONG) - input
Indicates the type of exit being called. The cluster workload exit is the only supported exit.

- The ExitId value must be MQXT_CLUSTER_WORKLOAD_EXIT

ExitReason (MQLONG) - input
Indicates the reason for invoking the cluster workload exit. ExitReason takes one of the following values:

MQXR_INIT
Indicates that the exit is being invoked for the first time.

Acquire and initialize any resources that the exit might need, such as main storage.

MQXR_TERM
Indicates that the exit is about to be terminated.

Free any resources that the exit might have acquired since it was initialized, such as main storage.

MQXR_CLWL_OPEN
Called by MQOPEN.

MQXR_CLWL_PUT
Called by MQPUT or MQPUT1.

MQXR_CLWL_MOVE
Called by MCA when the channel state has changed.

MQXR_CLWL_REPOS
Called by MQPUT or MQPUT1 for a repository-manager PCF message.

MQXR_CLWL_REPOS_MOVE
Called by MCA for a repository-manager PCF message if the channel state has changed.

ExitResponse (MQLONG) - output
Set ExitResponse to indicate whether processing of the message continues. It must be one of the following values:

MQXCC_OK
Continue processing the message normally.

- DestinationChosen identifies the destination to which the message is to be sent.

MQXCC_SUPPRESS_FUNCTION
Discontinue processing the message.

- The actions taken by the queue manager depend on the reason the exit was invoked:

Table 24. Actions taken by the queue manager

ExitReason	Action taken
• MQXR_CLWL_OPEN • MQXR_CLWL_REPOS • MQXR_CLWL_PUT	MQOPEN, MQPUT, or MQPUT1 call fail with completion code MQCC_FAILED and reason code MQRC_STOPPED_BY_CLUSTER_EXIT.
• MQXR_CLWL_MOVE • MQXR_CLWL_REPOS_MOVE	The message is placed on the dead-letter queue.

MQXCC_SUPPRESS_EXIT
Continue processing the current message normally. Do not invoke the exit again until the queue manager shuts down.

The queue manager processes subsequent messages as if the ClusterWorkloadExit queue-manager attribute is blank. DestinationChosen identifies the destination to which the current message is sent.

Any other value
Process the message as if MQXCC_SUPPRESS_FUNCTION is specified.

ExitResponse2 (MQLONG) - input/output
Set ExitResponse2 to provide the queue manager with more information.

- MQXR2_STATIC_CACHE is the default value, and is set on entry to the exit.
- When ExitReason has the value MQXR_INIT, the exit can set one of the following values in ExitResponse2:

  MQXR2_STATIC_CACHE
  The exit requires a static cluster cache.
  - If the cluster cache is static, the exit need not use the MQXCLWLN call to navigate the chains of records in the cluster cache.
  - If the cluster cache is dynamic, the exit cannot navigate correctly through the records in the cache.

  Note: The queue manager processes the return from the MQXR_INIT call as though the exit had returned MQXCC_SUPPRESS_EXIT in the ExitResponse field.

  MQXR2_DYNAMIC_CACHE
  The exit can operate with either a static or dynamic cache.
  - If the exit returns this value, the exit must use the MQXCLWLN call to navigate the chains of records in the cluster cache.

Feedback (MQLONG) - input
A reserved field. The value is zero.

Flags (MQLONG) - input
Indicates information about the message being put.

- The value of Flags is MQWXP_PUT_BY_CLUSTER_CHL. The message originates from a cluster channel, rather than locally or from a non-cluster channel. In other words, the message has come from another cluster queue manager.

Reserved (MQLONG) - input
A reserved field. The value is zero.

ExitUserArea (MQBYTE16) - input/output
Set ExitUserArea to communicate between calls to the exit.

- ExitUserArea is initialized to binary zero before the first invocation of the exit. Any changes made to this field by the exit are preserved across the invocations of the exit that occur between the MQCONN call and the matching MQDISC call. The field is reset to binary zero when the MQDISC call occurs.
- The first invocation of the exit is indicated by the ExitReason field having the value MQXR_INIT.
- The following constants are defined:

  MQXUA_NONE - string
  MQXUA_NONE_ARRAY - character array
  No user information. Both constants are binary zero for the length of the field.

  MQ_EXIT_USER_AREA_LENGTH
  The length of ExitUserArea.

ExitData (MQCHAR32) - input
The value of the ClusterWorkloadData queue-manager attribute. If no value has been defined for that attribute, this field is all blanks.

- The length of ExitData is given by MQ_EXIT_DATA_LENGTH.

MsgDescPtr (PMQMD) - input
The address of a copy of the message descriptor (MQMD) for the message being processed.

- Any changes made to the message descriptor by the exit are ignored by the queue manager.
- If ExitReason has one of the following values MsgDescPtr is set to the null pointer, and no message descriptor is passed to the exit:
  - MQXR_INIT
  - MQXR_TERM
  - MQXR_CLWL_OPEN

MsgBufferPtr (PMQVOID) - input
The address of a buffer containing a copy of the first MsgBufferLength bytes of the message data.

- Any changes made to the message data by the exit are ignored by the queue manager.
- No message data is passed to the exit when:
  - MsgDescPtr is the null pointer.
  - The message has no data.
  - The ClusterWorkloadLength queue-manager attribute is zero.

  In these cases, MsgBufferPtr is the null pointer.

MsgBufferLength (MQLONG) - input
The length of the buffer containing the message data passed to the exit.

- The length is controlled by the ClusterWorkloadLength queue-manager attribute.
- The length might be less than the length of the complete message, see MsgLength.

MsgLength (MQLONG) - input
The length of the complete message passed to the exit.

- MsgBufferLength might be less than the length of the complete message.
- MsgLength is zero if ExitReason is MQXR_INIT, MQXR_TERM, or MQXR_CLWL_OPEN.

QName (MQCHAR48) - input
The name of the destination queue. The queue is a cluster queue.

- The length of QName is MQ_Q_NAME_LENGTH.

QMgrName (MQCHAR48) - input
The name of the local queue manager that has invoked the cluster workload exit.

- The length of QMgrName is MQ_Q_MGR_NAME_LENGTH.

DestinationCount (MQLONG) - input
The number of possible destinations. Destinations are instances of the destination queue and are described by destination records.

- A destination record is a MQWDR structure. There is one structure for each possible route to each instance of the queue.
- MQWDR structures are addressed by an array of pointers, see DestinationArrayPtr.

DestinationChosen (MQLONG) - input/output
The chosen destination.

- The number of the MQWDR structure that identifies the route and queue instance where the message is to be sent.
- The value is in the range 1 - DestinationCount.
- On input to the exit, DestinationChosen indicates the route and queue instance that the queue manager has selected. The exit can accept this choice, or choose a different route and queue instance.
- The value set by the exit must be in the range 1 - DestinationCount. If any other value is returned, the queue manager uses the value of DestinationChosen on input to the exit.

DestinationArrayPtr (PPMQWDR) - input
The address of an array of pointers to destination records (MQWDR).

- There are DestinationCount destination records.

QArrayPtr (PPMQWQR) - input
The address of an array of pointers to queue records (MQWQR).

- If queue records are available, there are DestinationCount of them.
- If no queue records are available, QArrayPtr is the null pointer.

  Note: QArrayPtr can be the null pointer even when DestinationCount is greater than zero.

CacheContext (MQPTR) : Version 2 - input
The CacheContext field is reserved for use by the queue manager. The exit must not alter the value of this field.

CacheType (MQLONG) : Version 2 - input
The cluster cache has one of the following types:

MQCLCT_STATIC
The cache is static.

- The size of the cache is fixed, and cannot grow as the queue manager operates.
- You do not need to use the MQXCLWLN call to navigate the records in this type of cache.

MQCLCT_DYNAMIC
The cache is dynamic.

- The size of the cache can increase in order to accommodate the varying cluster information.
- You must use the MQXCLWLN call to navigate the records in this type of cache.

CLWLMRUChannels (MQLONG) : Version 3 - input
Indicates the maximum number of active outbound cluster channels, to be considered for use by the cluster workload choice algorithm.

- CLWLMRUChannels is a value 1 - 999 999 999.

pEntryPoints (PMQIEP) : Version 4
The address of an MQIEP structure through which MQI and DCI calls can be made.

Initial values and language declarations for MQWXP:

Initial values and C and High Level Assembler Language declarations for MQWXP - Cluster workload exit parameter structure.

Table 25. Initial values of fields in MQWXP

Field name	Name of constant	Value of constant
StrucId	MQWXP_STRUC_ID	'WXP? '
Version	MQWXP_VERSION_2	2
ExitId	None	0
ExitReason	MQXCC_OK	0
ExitResponse	None	0
ExitResponse2	None	0
Flags	None	0
ExitUserArea	{MQXUA_NONE_ARRAY}	0
ExitData	None	""
MsgDescPtr	None	NULL
MsgBufferPtr	None	NULL
MsgBufferLength	None	0
MsgBufferPtr	None	0
QName	None	""
QMgrName	None	""
DestinationCount	None	0
DestinationChosen	None	0
DestinationArrayPtr	None	NULL
QArrayPtr	None	NULL
CacheContext	None	NULL
CacheType	MQCLCT_DYNAMIC	1
CLWLMRUChannels	None	0
pEntryPoints	None	NULL

Notes:

1. The symbol ? represents a single blank character.
2. In the C programming language, the macro variable MQWXP_DEFAULT contains the default values. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQWDR MyWXP = {MQWXP_DEFAULT};
   ```

C declaration

```
typedef struct tagMQWXP {
  MQCHAR4   StrucId;              /* Structure identifier */
  MQLONG    Version;              /* Structure version number */
  MQLONG    ExitId;               /* Type of exit */
  MQLONG    ExitReason;           /* Reason for invoking exit */
  MQLONG    ExitResponse;         /* Response from exit */
  MQLONG    ExitResponse2;        /* Reserved */
  MQLONG    Feedback;             /* Reserved */
  MQLONG    Flags;                /* Flags */
```

```
  MQBYTE16  ExitUserArea;         /* Exit user area */
  MQCHAR32  ExitData;             /* Exit data */
  PMQMD     MsgDescPtr;           /* Address of message descriptor */
  PMQVOID   MsgBufferPtr;         /* Address of buffer containing some
                                     or all of the message data */
  MQLONG    MsgBufferLength;      /* Length of buffer containing message
                                     data */
  MQLONG    MsgLength;            /* Length of complete message */
  MQCHAR48  QName;                /* Queue name */
  MQCHAR48  QMgrName;             /* Name of local queue manager */
  MQLONG    DestinationCount;     /* Number of possible destinations */
  MQLONG    DestinationChosen;    /* Destination chosen */
  PPMQWDR   DestinationArrayPtr;  /* Address of an array of pointers to
                                     destination records */
  PPMQWQR   QArrayPtr;            /* Address of an array of pointers to
                                     queue records */
  /* version 1 */
  MQPTR     CacheContext;         /* Context information */
  MQLONG    CacheType;            /* Type of cluster cache */
  /* version 2 */
  MQLONG    CLWLMRUChannnels;     /* Maximum number of most recently
                                     used cluster channels */
  /* version 3 */
  PMQIEP    pEntryPoints;         /* Address of the MQIEP structure */
  /* version 4 */
};
```

High Level Assembler

```
MQWXP                       DSECT
MQWXP_STRUCID               DS   CL4      Structure identifier
MQWXP_VERSION               DS   F        Structure version number
MQWXP_EXITID                DS   F        Type of exit
MQWXP_EXITREASON            DS   F        Reason for invoking exit
MQWXP_EXITRESPONSE          DS   F        Response from exit
MQWXP_EXITRESPONSE2         DS   F        Reserved
MQWXP_FEEDBACK              DS   F        Reserved
MQWXP_RESERVED              DS   F        Reserved
MQWXP_EXITUSERAREA          DS   XL16     Exit user area
MQWXP_EXITDATA              DS   CL32     Exit data
MQWXP_MSGDESCPTR            DS   F        Address of message
*                                         descriptor
MQWXP_MSGBUFFERPTR          DS   F        Address of buffer containing
*                                         some or all of the message
*                                         data
MQWXP_MSGBUFFERLENGTH       DS   F        Length of buffer containing
*                                         message data
MQWXP_MSGLENGTH             DS   F        Length of complete message
MQWXP_QNAME                 DS   CL48     Queue name
MQWXP_QMGRNAME              DS   CL48     Name of local queue manager
MQWXP_DESTINATIONCOUNT      DS   F        Number of possible
*                                         destinations
MQWXP_DESTINATIONCHOSEN     DS   F        Destination chosen
MQWXP_DESTINATIONARRAYPTR   DS   F        Address of an array of
*                                         pointers to destination
*                                         records
MQWXP_QARRAYPTR             DS   F        Address of an array of
*                                         pointers to queue records
MQWXP_CACHECONTEXT          DS   F        Context information
MQWXP_CACHETYPE             DS   F        Type of cluster cache
MQWXP_CLWLMRUCHANNELS       DS   F        Number of most recently used
*                                         channels for workload balancing

MQWXP_LENGTH                EQU  *-MQWXP  Length of structure
                            ORG  MQWXP
MQWXP_AREA                  DS   CL(MQWXP_LENGTH)
```

MQWDR - Cluster workload destination record structure:

The following table summarizes the fields in the MQWDR - Cluster workload destination record structure.

Table 26. Fields in MQWDR

Field	Description	Page
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQWDR structure	StrucLength
QMgrFlags	Queue-manager flags	QMgrFlags
QMgrIdentifier	Queue-manager identifier	QMgrIdentifier
QMgrName	Queue-manager name	QMgrName
ClusterRecOffset	Logical offset of first cluster record (MQWCR)	ClusterRecOffset
ChannelState	Channel state	ChannelState
ChannelDefOffset	Logical offset of channel-definition structure (MQCD)	ChannelDefOffset
Note: The remaining fields are ignored if Version is less than MQWDR_VERSION_2.		
DestSeqNumber	Channel destination sequence number	DestSeqNumber
DestSeqFactor	Channel destination sequence factor for weighting	DestSeqFactor

The cluster workload destination record structure contains information relating to one of the possible destinations for the message. There is one cluster workload destination record structure for each instance of the destination queue.

The cluster workload destination record structure is supported in all environments.

Additionally, the MQWDR1 and MQWDR2 structures are available for backwards compatibility.

Fields in MQWDR - Cluster workload destination record structure:

Description of the parameters in the MQWDR - Cluster workload destination record structure.

StrucId (MQCHAR4) - input
The structure identifier for the cluster workload destination record structure.

- The StrucId value is MQWDR_STRUC_ID.
- For the C programming language, the constant MQWDR_STRUC_ID_ARRAY is also defined. It has the same value as MQWDR_STRUC_ID. It is an array of characters instead of a string.

Version (MQLONG) - input
The structure version number. Version takes one of the following values:

MQWDR_VERSION_1
Version-1 cluster workload destination record.

MQWDR_VERSION_2
Version-2 cluster workload destination record.

MQWDR_CURRENT_VERSION
Current® version of cluster workload destination record.

StrucLength (MQLONG) - input
The length of MQWDR structure. StrucLength takes one of the following values:

MQWDR_LENGTH_1
Length of version-1 cluster workload destination record.

MQWDR_LENGTH_2
Length of version-2 cluster workload destination record.

MQWDR_CURRENT_LENGTH
Length of current version of cluster workload destination record.

QMgrFlags (MQLONG) - input
Queue manager flags indicating properties of the queue manager that hosts the instance of the destination queue described by the MQWDR structure. The following flags are defined:

MQQMF_REPOSITORY_Q_MGR
Destination is a full repository queue manager.

MQQMF_CLUSSDR_USER_DEFINED
Cluster-sender channel was defined manually.

MQQMF_CLUSSDR_AUTO_DEFINED
Cluster-sender channel was defined automatically.

MQQMF_AVAILABLE
Destination queue manager is available to receive messages.

Other values
Other flags in the field might be set by the queue manager for internal purposes.

QMgrIdentifier (MQCHAR48) - input
The queue manager identifier is a unique identifier for the queue manager that hosts the instance of the destination queue described by the MQWDR structure.

- The identifier is generated by the queue manager.
- The length of QMgrIdentifier is MQ_Q_MGR_IDENTIFIER_LENGTH.

QMgrName (MQCHAR48) - input
The name of the queue manager that hosts the instance of the destination queue described by the MQWDR structure.

- QMgrName can be the name of the local queue manager, as well another queue manager in the cluster.
- The length of QMgrName is MQ_Q_MGR_NAME_LENGTH.

ClusterRecOffset (MQLONG) - input
The logical offset of the first MQWCR structure that belongs to the MQWDR structure.

- For static caches, ClusterRecOffset is the offset of the first MQWCR structure that belongs to the MQWDR structure.
- The offset is measured in bytes from the start of the MQWDR structure.
- Do not use the logical offset for pointer arithmetic with dynamic caches. To obtain the address of the next record, the MQXCLWLN call must be used.

ChannelState (MQLONG) - input
The state of the channel that links the local queue manager to the queue manager identified by the MQWDR structure. The following values are possible:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_INACTIVE
Channel is not active.

MQCHS_INITIALIZING
Channel is initializing.

MQCHS_PAUSED
Channel has paused.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_STOPPING
Channel is stopping.

MQCHS_STOPPED
Channel has stopped.

ChannelDefOffset (MQLONG) - input
The logical offset of the channel definition (MQCD) for the channel that links the local queue manager to the queue manager identified by the MQWDR structure.

- ChannelDefOffset is like ClusterRecOffset
- The logical offset cannot be used in pointer arithmetic. To obtain the address of the next record, the MQXCLWLN call must be used.

DestSeqFactor (MQLONG) - input
The destination sequence factor that allows a choice of the channel based on weight.

- DestSeqFactor is used before the queue manager changes it.
- The workload manager increases DestSeqFactor in a way that ensures messages are distributed down channels according to their weight.

DestSeqNumber (MQLONG) - input
The cluster channel destination value before the queue manager changes it.

- The workload manager increases DestSeqNumber every time a message is put down that channel.
- Workload exits can use DestSeqNumber to decide which channel to put a message down.

Initial values and language declarations for MQWDR:

Initial values and C and High Level Assembler Language declarations for MQWDR - Cluster workload destination record.

Table 27. Initial values of fields in MQWDR

Field name	Name of constant	Value of constant
StrucId	MQWDR_STRUC_ID	'WDR? '
Version	MQWDR_VERSION_1	1
StrucLength	MQWDR_CURRENT_LENGTH [3]	136
QMgrFlags	MQWDR_NONE	0
QMgrIdentifier	None	""
QMgrName	None	""
ClusterRecOffset	None	0
ChannelState	None	0
ChannelDefOffset	None	0
DestSeqNumber	None	0
DestSeqFactor	None	0

Table 27. Initial values of fields in MQWDR (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ? represents a single blank character. 2. In the C programming language, the macro variable MQWDR_DEFAULT contains the default values. Use it in the following way to provide initial values for the fields in the structure: `MQWDR MyWDR = {MQWDR_DEFAULT};` 3. The initial values intentionally set the length of the structure to the length of the current version, and not version 1 of the structure.		

High Level Assembler

```
MQWDR                       DSECT
MQWDR_STRUCID               DS   CL4      Structure identifier
MQWDR_VERSION               DS   F        Structure version number
MQWDR_STRUCLENGTH           DS   F        Length of MQWDR structure
MQWDR_QMGRFLAGS             DS   F        Queue-manager flags
MQWDR_QMGRIDENTIFIER        DS   CL48     Queue-manager identifier
MQWDR_QMGRNAME              DS   CL48     Queue-manager name
MQWDR_CLUSTERRECOFFSET      DS   F        Offset of first cluster
*                                         record
MQWDR_CHANNELSTATE          DS   F        Channel state
MQWDR_CHANNELDEFOFFSET      DS   F        Offset of channel definition
*                                         structure
MQWDR_LENGTH                EQU  *-MQWDR  Length of structure
                            ORG  MQWDR
MQWDR_AREA                  DS   CL(MQWDR_LENGTH)
```

C declaration

```
typedef struct tagMQWDR {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    StrucLength;       /* Length of MQWDR structure */
  MQLONG    QMgrFlags;         /* Queue-manager flags */
  MQCHAR48  QMgrIdentifier;    /* Queue-manager identifier */
  MQCHAR48  QMgrName;          /* Queue-manager name */
  MQLONG    ClusterRecOffset;  /* Offset of first cluster record */
  MQLONG    ChannelState;      /* Channel state */
  MQLONG    ChannelDefOffset;  /* Offset of channel definition structure */
  /* Ver:1 */
  MQLONG    DestSeqNumber;     /* Cluster channel destination sequence number */
  MQINT64   DestSeqFactor;     /* Cluster channel factor sequence number */
  /* Ver:2 */
};
```

MQWQR - Cluster workload queue record structure:

The following table summarizes the fields in the MQWQR - Cluster workload queue record structure.

Table 28. Fields in MQWQR

Field	Description	Page
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQWQR structure	StrucLength
QFlags	Queue flags	QFlags
QName	Queue name	QName
QMgrIdentifier	Queue-manager identifier	QMgrIdentifier
ClusterRecOffset	Offset of first cluster record (MQWCR)	ClusterRecOffset
QType	Queue type	QType
QDesc	Queue description	QDesc
DefBind	Default binding	DefBind
DefPersistence	Default message persistence	DefPersistence
DefPriority	Default message priority	DefPriority
InhibitPut	Whether put operations on the queue are allowed	InhibitPut
Note: The remaining fields are ignored if Version is less than MQWQR_VERSION_2.		
CWLQueuePriority	A value 0 - 9 representing the priority of the queue	CLWLQueuePriority
CLWLQueueRank	A value 0 - 9 representing the rank of the queue	CLWLQueueRank
Note: The remaining fields are ignored if Version is less than MQWQR_VERSION_3.		
DefPutResponse	Default put response	DefPutResponse

The cluster workload queue record structure contains information relating to one of the possible destinations for the message. There is one cluster workload queue record structure for each instance of the destination queue.

The cluster workload queue record structure is supported in all environments.

Additionally, the MQWQR1 and MQWQR2 structures are available for backwards compatibility.

Fields in MQWQR - Cluster workload queue record structure:

Description of the fields in the MQWQR - Cluster workload queue record structure.

StrucId (MQCHAR4) - input
The structure identifier for the cluster workload queue record structure.

- The StrucId value is MQWQR_STRUC_ID.
- For the C programming language, the constant MQWQR_STRUC_ID_ARRAY is also defined. It has the same value as MQWQR_STRUC_ID. It is an array of characters instead of a string.

Version (MQLONG) - input
The structure version number. Version takes one of the following values:

MQWQR_VERSION_1
Version-1 cluster workload queue record.

MQWQR_VERSION_2
Version-2 cluster workload queue record.

MQWQR_VERSION_3
Version-3 cluster workload queue record.

MQWQR_CURRENT_VERSION
Current version of cluster workload queue record.

StrucLength (MQLONG) - input
The length of MQWQR structure. StrucLength takes one of the following values:

MQWQR_LENGTH_1
Length of version-1 cluster workload queue record.

MQWQR_LENGTH_2
Length of version-2 cluster workload queue record.

MQWQR_LENGTH_3
Length of version-3 cluster workload queue record.

MQWQR_CURRENT_LENGTH
Length of current version of cluster workload queue record.

QFlags (MQLONG) - input
The queue flags indicate properties of the queue. The following flags are defined:

MQQF_LOCAL_Q
Destination is a local queue.

MQQF_CLWL_USEQ_ANY
Allow use of local and remote queues in puts.

MQQF_CLWL_USEQ_LOCAL
Allow only local queue puts.

Other values
Other flags in the field might be set by the queue manager for internal purposes.

QName (MQCHAR48) - input
The name of the queue that is one of the possible destinations of the message.

- The length of QName is MQ_Q_NAME_LENGTH.

QMgrIdentifier (MQCHAR48) - input
The queue manager identifier is a unique identifier for the queue manager that hosts the instance of the queue described by the MQWQR structure.

- The identifier is generated by the queue manager.
- The length of QMgrIdentifier is MQ_Q_MGR_IDENTIFIER_LENGTH.

ClusterRecOffset (MQLONG) - input
The logical offset of the first MQWCR structure that belongs to the MQWQR structure.

- For static caches, ClusterRecOffset is the offset of the first MQWCR structure that belongs to the MQWQR structure.
- The offset is measured in bytes from the start of the MQWQR structure.
- Do not use the logical offset for pointer arithmetic with dynamic caches. To obtain the address of the next record, the MQXCLWLN call must be used.

QType (MQLONG) - input
The queue type of the destination queue. The following values are possible:

MQCQT_LOCAL_Q
Local queue.

MQCQT_ALIAS_Q
Alias queue.

MQCQT_REMOTE_Q
Remote queue.

MQCQT_Q_MGR_ALIAS
Queue-manager alias.

QDesc (MQCHAR64) - input
The queue description queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure.

- The length of QDesc is MQ_Q_DESC_LENGTH.

DefBind (MQLONG) - input
The default binding queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. Either MQBND_BIND_ON_OPEN or MQBND_BIND_ON_GROUP must be specified when using groups with clusters. The following values are possible:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

DefPersistence (MQLONG) - input
The default message persistence queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. The following values are possible:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority (MQLONG) - input

The default message priority queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. The priority range is 0 - MaxPriority.

- 0 is the lowest priority.
- MaxPriority is the queue manager attribute of the queue manager that hosts this instance of the destination queue.

InhibitPut (MQLONG) - input
The put inhibited queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. The following values are possible:

MQQA_PUT_INHIBITED
Put operations are inhibited.

MQQA_PUT_ALLOWED
Put operations are allowed.

CLWLQueuePriority (MQLONG) - input
The cluster workload queue priority attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure.

CLWLQueueRank (MQLONG) - input
The cluster workload queue rank defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure.

DefPutResponse (MQLONG) - input
The default put response queue attribute defined on the queue manager that hosts the instance of the destination queue described by the MQWQR structure. The following values are possible:

MQPRT_SYNC_RESPONSE
Synchronous response to MQPUT or MQPUT1 calls.

MQPRT_ASYNC_RESPONSE
Asynchronous response to MQPUT or MQPUT1 calls.

Initial values and language declarations for MQWQR:

Initial values and C and High Level Assembler Language declarations for MQWQR - Cluster workload queue record.

Table 29. Initial values of fields in MQWQR

Field name	Name of constant	Value of constant
StrucId	MQWQR_STRUC_ID_ARRAY	'WQR? '
Version	MQWQR_VERSION_1	1
StrucLength	MQWQR_CURRENT_LENGTH [3]	212
QFlags	None	0
QName	None	""
QMgrIdentifier	None	""
ClusterRecOffset	None	0
QType	None	0
QDesc	None	""
DefBind	None	0
DefPersistence	None	0
DefPriority	None	0
InhibitPut	None	0
CLWLQueuePriority	None	0
CLWLQueueRank	None	0
DefPutResponse	None	1

Notes:

1. The symbol ? represents a single blank character.
2. In the C programming language, the macro variable MQWQR_DEFAULT contains the default values. Use it in the following way to provide initial values for the fields in the structure:
   ```
   MQWQR MyWQR = {MQWQR_DEFAULT};
   ```
3. The initial values intentionally set the length of the structure to the length of the current version, and not version 1 of the structure.

C declaration

```
typedef struct tagMQWQR {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    StrucLength;       /* Length of MQWQR structure */
  MQLONG    QFlags;            /* Queue flags */
  MQCHAR48  QName;             /* Queue name */
  MQCHAR48  QMgrIdentifier;    /* Queue-manager identifier */
  MQLONG    ClusterRecOffset;  /* Offset of first cluster record */
  MQLONG    QType;             /* Queue type */
  MQCHAR64  QDesc;             /* Queue description */
  MQLONG    DefBind;           /* Default binding */
  MQLONG    DefPersistence;    /* Default message persistence */
  MQLONG    DefPriority;       /* Default message priority */
  MQLONG    InhibitPut;        /* Whether put operations on the queue
```

```
                                  are allowed */
  /* version 2 */
  MQLONG    CLWLQueuePriority; /* Queue priority */
  MQLONG    CLWLQueueRank;     /* Queue rank */
  /* version 3 */
  MQLONG    DefPutResponse;    /* Default put response */
};
```

High Level Assembler

```
MQWQR                     DSECT
MQWQR_STRUCID             DS   CL4      Structure identifier
MQWQR_VERSION             DS   F        Structure version number
MQWQR_STRUCLENGTH         DS   F        Length of MQWQR structure
MQWQR_QFLAGS              DS   F        Queue flags
MQWQR_QNAME               DS   CL48     Queue name
MQWQR_QMGRIDENTIFIER      DS   CL48     Queue-manager identifier
MQWQR_CLUSTERRECOFFSET    DS   F        Offset of first cluster
*                                       record
MQWQR_QTYPE               DS   F        Queue type
MQWQR_QDESC               DS   CL64     Queue description
MQWQR_DEFBIND             DS   F        Default binding
MQWQR_DEFPERSISTENCE      DS   F        Default message persistence
MQWQR_DEFPRIORITY         DS   F        Default message priority
MQWQR_INHIBITPUT          DS   F        Whether put operations on
*                                       the queue are allowed
MQWQR_DEFPUTRESPONSE      DS   F        Default put response
MQWQR_LENGTH              EQU  *-MQWQR  Length of structure
                          ORG  MQWQR
MQWQR_AREA                DS   CL(MQWQR_LENGTH)
```

MQWCR - Cluster workload cluster record structure:

The following table summarizes the fields in the MQWCR cluster workload record structure.

Table 30. Fields in MQWCR

Field	Description	Page
ClusterName	Name of cluster	ClusterName
ClusterRecOffset	Offset of next cluster record (MQWCR)	ClusterRecOffset
ClusterFlags	Cluster flags	ClusterFlags

The cluster workload cluster record structure contains information about a cluster. For each cluster the destination queue belongs to, there is one cluster workload cluster record structure.

The cluster workload cluster record structure is supported in all environments.

Fields in the MQWCR - Cluster workload cluster record structure.:

Description of the fields in the MQWCR - Cluster workload cluster record structure.

ClusterName (MQCHAR48) - input
The name of a cluster to which the instance of the destination queue that owns the MQWCR structure belongs. The destination queue instance is described by an MQWDR structure.

- The length of ClusterName is MQ_CLUSTER_NAME_LENGTH.

ClusterRecOffset (MQLONG) - input
The logical offset of the next MQWCR structure.

- If there are no more MQWCR structures, ClusterRecOffset is zero.
- The offset is measured in bytes from the start of the MQWCR structure.

ClusterFlags (MQLONG) - input
The cluster flags indicate properties of the queue manager identified by the MQWCR structure. The following flags are defined:

MQQMF_REPOSITORY_Q_MGR
Destination is a full repository queue manager.

MQQMF_CLUSSDR_USER_DEFINED
Cluster-sender channel was defined manually.

MQQMF_CLUSSDR_AUTO_DEFINED
Cluster-sender channel was defined automatically.

MQQMF_AVAILABLE
Destination queue manager is available to receive messages.

Other values
Other flags in the field might be set by the queue manager for internal purposes.

Initial values and language declarations for MQWCR:

Initial values and C and High Level Assembler Language declarations for MQWCR - Cluster workload cluster record structure.

Table 31. Initial values of fields in MQWCR

Field name	Name of constant	Value of constant
ClusterName	None	""
ClusterRecOffset	None	0
ClusterFlags	None	0

C declaration

```
typedef struct tagMQWCR {
  MQCHAR48  ClusterName;       /* Cluster name */
  MQLONG    ClusterRecOffset;  /* Offset of next cluster record */
  MQLONG    ClusterFlags;      /* Cluster flags */
};
```

High Level Assembler

```
MQWCR                          DSECT
MQWCR_CLUSTERNAME              DS   CL48     Cluster name
MQWCR_CLUSTERRECOFFSET         DS   F        Offset of next cluster
*                                           record
MQWCR_CLUSTERFLAGS             DS   F        Cluster flags
MQWCR_LENGTH                   EQU  *-MQWCR  Length of structure
                               ORG  MQWCR
MQWCR_AREA                     DS   CL(MQWCR_LENGTH)
```

Channel programs

This section looks at the different types of channel programs (MCAs) available for use at the channels.

The names of the MCAs are shown in the following tables.

Table 32. Channel programs for Windows, UNIX and Linux systems

Program name	Direction of connection	Communication
amqrmppa		Any
runmqlsr	Inbound	Any
amqcrs6a	Inbound	LU 6.2
amqcrsta	Inbound	TCP
runmqchl	Outbound	Any
runmqchi	Outbound	Any

runmqlsr (Run WebSphere MQ listener), runmqchl (Run WebSphere MQ channel), and runmqchi (Run WebSphere MQ channel initiator) are control commands that you can enter at the command line.

amqcrsta is invoked for TCP channels on UNIX and Linux systems using inetd, where no listener is started.

amqcrs6a is invoked as a transaction program when using LU6.2

Environment Variables

A list of all the server and client Environment Variables. Example of use, on UNIX and Linux systems use: `export [environment variable]=filename`. On Windows Systems, use: `Set [environment variable]=filename`.

AMQ_MQS_INI_LOCATION
On UNIX and Linux systems, you can alter the location used for the `mqs.ini` file by setting the location of the `mqs.ini` file in this variable. This variable must be set at the system level.

AMQ_SSL_ALLOW_DEFAULT_CERT
When the `AMQ_SSL_ALLOW_DEFAULT_CERT` environment variable is not set, an application can connect to a queue manager with a personal certificate in the client keystore only when the certificate includes the label name of `ibmwebspheremq<userid>`. When the `AMQ_SSL_ALLOW_DEFAULT_CERT` environment variable is set, the certificate does not require the label name of `ibmwebspheremq<userid>`. That is, the certificate that is used to connect to a queue manager can be a default certificate, provided that a default certificate is present in the queue repository, and the key repository does not contain a personal certificate with the prefix `ibmwebspheremq<userid>`. For more information, see the technote Specifying the userid in the SSL certificate label for an MQ client.

A value of 1 enables the use of a default certificate.

GMQ_MQ_LIB
When both the IBM WebSphere MQ MQI client and IBM WebSphere MQ server are installed on your system, MQAX applications run against the server by default. To run MQAX against the client, the client bindings library must be specified in the GMQ_MQ_LIB environment variable, for example, set `GMQ_MQ_LIB=mqic.dll`. For a client only installation, it is not necessary to set the GMQ_MQ_LIB environment variable. When this variable is not set, WebSphere MQ attempts to load `amqzst.dll`. If this DLL is not present (as is the case in a client only installation), WebSphere MQ attempts to load `mqic.dll`.

HOME
This variable contains the name of the directory which is searched for the `mqclient.ini` file. This file contains configuration information used by IBM WebSphere MQ MQI clients on UNIX and Linux systems.

HOMEDRIVE and HOMEPATH
To be used both of these variables must be set. They are used to contain the name of the

directory which is searched for the mqclient.ini file. This file contains configuration information used by IBM WebSphere MQ MQI clients on Windows systems.

LDAP_BASEDN
The required environment variable for running an LDAP sample program. It specifies the base Distinguished Name for the directory search.

LDAP_HOST
An optional variable for running an LDAP sample program. It specifies the name of the host where the LDAP server is running; it defaults to the local host if it is not specified

LDAP_VERSION
An optional variable for running an LDAP sample program. It specifies the version of the LDAP protocol to be used, and can be either 2 or 3. Most LDAP servers now support version 3 of the protocol; they all support the older version 2. This sample works equally well with either version of the protocol, and if it is not specified it defaults to version 2.

MQAPI_TRACE_LOGFILE
The sample API exit program generates an MQI trace to a user-specified file with a prefix defined in the MQAPI_TRACE_LOGFILE environment variable.

MQCCSID
Specifies the coded character set number to be used and overrides the native CCSID of the application.

MQCERTVPOL
Determines the type of certificate validation used:

ANY Use any certificate validation policy supported by the underlying secure sockets library. This setting is the default setting.

RFC5280
Use only certificate validation which complies with the RFC 5280 standard.

MQCHLLIB
Specifies the directory path to the file containing the client channel definition table (CCDT). The file is created on the server, but can be copied across to the WebSphere MQ MQI client workstation.

MQCHLTAB
MQCHLTAB specifies the name of the file containing the client channel definition table (ccdt). The default file name is AMQCLCHL.TAB.

MQC_IPC_HOST
When sharing IBM WebSphere MQ files and the generated value of myHostName creates a problem set myHostName using the environment variable MQC_IPC_HOST

MQCLNTCF
Use this environment variable to modify the mqclient.ini file path.

MQ_CONNECT_TYPE
On IBM WebSphere MQ for Windows, UNIX and Linux systems, use this environment variable in combination with the type of binding specified in the Options field of the MQCNO structure used on an MQCONNX call. See MQCONNX environment variable

MQ_FILE_PATH
During the installation of the runtime package on the Windows platform, a new environment variable called MQ_FILE_PATH is configured. This environment variable contains the same data as the following key in the Windows Registry:

HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation\<InstallationName>\FilePath

MQIPADDRV
MQIPADDRV specifies which IP protocol to use for a channel connection. It has the possible

string values of "MQIPADDR_IPV4" or "MQIPADDR_IPV6". These values have the same meanings as IPV4 and IPV6 in ALTER QMGR IPADDRV. If it is not set, "MQIPADDR_IPV4" is assumed.

MQ_JAVA_DATA_PATH
Specifies the directory for log and trace output.

MQ_JAVA_INSTALL_PATH
Specifies the directory where IBM WebSphere MQ classes for Java are installed, as shown in IBM WebSphere MQ classes for Java installation directories.

MQ_JAVA_LIB_PATH
Specifies the directory where the IBM WebSphere MQ classes for Java libraries are stored. Some scripts supplied with IBM WebSphere MQ classes for Java, such as IVTRun, use this environment variable.

MQNAME
MQNAME specifies the local NetBIOS name that the IBM WebSphere MQ processes can use.

MQNOREMPOOL
When you set this variable, it switches off channel pooling and causes channels to run as threads of the listener.

MQPSE_TRACE_LOGFILE
Use when you Publish the Exit Sample Program. In the application process to be traced, this environment variable describes where the trace files must be written to. See The Publish Exit sample program

MQSERVER
MQSERVER environment variable is used to define a minimal channel. You cannot use MQSERVER to define an SSL channel or a channel with channel exits. MQSERVER specifies the location of the WebSphere MQ server and the communication method to be used.

MQ_SET_NODELAYACK
When you set this variable, it switches off TCP delayed acknowledgment

When you set this variable on AIX, the setting switches off TCP delayed acknowledgment by calling the operating system's setsockopt call with the TCP_NODELAYACK option. Only AIX supports this function, so the MQ_SET_NODELAYACK environment variable only has an effect on AIX.

MQSNOAUT
MQSNOAUT disables the object authority manager (OAM) and prevents any security checking. The MQSNOAUT variable only takes effect when a queue manager is created.

MQSPREFIX
As an alternative to changing the default prefix, you can use the environment variable MQSPREFIX to override the `DefaultPrefix` for the **crtmqm** command.

MQSSLCRYP
MQSSLCRYP holds a parameter string that you can use to configure the cryptographic hardware present on the system. The permitted values are the same as for the SSLCRYP parameter of the ALTER QMGR command.

MQSSLFIPS
MQSSLFIPS specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM WebSphere MQ. The values are the same as for the SSLFIPS parameter of the ALTER QMGR command.

MQSSLKEYR
MQSSLKEYR specifies the location of the key repository that holds the digital certificate

belonging to the user, in stem format. Stem format means that it includes the full path and the file name without an extension. For full details, see the SSLKEYR parameter of the ALTER QMGR command.

MQSSLPROXY
MQSSLPROXY specifies the host name and port number of the HTTP proxy server to be used by GSKit for OCSP checks.

MQSSLRESET
MQSSLRESET represents the number of unencrypted bytes sent and received on an SSL channel before the SSL secret key is renegotiated.

MQS_TRACE_OPTIONS
Use the environment variable MQS_TRACE_OPTIONS to activate the high detail and parameter tracing functions individually.

MQTCPTIMEOUT
This variable specifies how long IBM WebSphere MQ waits for a TCP connect call.

MQSUITEB
This variable specifies whether Suite B compliant cryptography is to be used. In the instance that Suite B cryptography is used you can specify the strength of the cryptography by setting MQSUITEB to one of the following:

- NONE
- 128_BIT, 192_BIT
- 128_BIT
- 192_BIT

ODQ_MSG
If you use a dead-letter queue handler that is different from RUNMQDLQ the source of the sample is available for you to use as your base. The sample is like the dead-letter handler provided within the product but trace and error reporting are different. Use the ODQ_MSG environment variable to set the name of the file containing error and information messages. The file provided is called amqsdlq.msg.

ODQ_TRACE
If you use a dead-letter queue handler that is different from RUNMQDLQ the source of the sample is available for you to use as your base. The sample is like the dead-letter handler provided within the product but trace and error reporting are different. Set the ODQ_TRACE environment variable to YES or yes to switch on tracing

OMQ_PATH
This environment variable is where you can find the First Failure Symptom report if your IBM WebSphere MQ automation classes for ActiveX script fails.

OMQ_TRACE
MQAX includes a trace facility to help the service organization identify what is happening when you have a problem. It shows the paths taken when you run your MQAX script. Unless you have a problem, run with tracing set off to avoid any unnecessary use of system resources.
OMQ_TRACE is one of the three environment variables set to control trace. Specifying any value for OMQ_TRACE switches the trace facility on. Even if you set OMQ_TRACE to OFF, trace is still active. See Using trace

OMQ_TRACE_PATH
One of the three environment variables set to control trace. See Using trace

OMQ_TRACE_LEVEL
One of the three environment variables set to control trace. See Using trace

ONCONFIG
The name of the Informix server configuration file. For example, on UNIX and Linux systems, use:

```
export ONCONFIG=onconfig.hostname_1
```

On Windows systems, use:

```
set ONCONFIG=onconfig.hostname_1
```

WCF_TRACE_ON
Two different trace methods are available for the WCF custom channel, the two trace methods are activated independently or together. Each method produces its own trace file, so when both trace methods have been activated, two trace output files are generated. There are four combinations for enabling and disabling the two different trace methods. As well as these combinations to enable WCF trace, the XMS .NET trace can also be enabled using the WCF_TRACE_ON environment variable. See WCF trace configuration and trace file names

WMQSOAP_HOME
Use when making additional configuration steps after the .NET SOAP over JMS service hosting environment is correctly installed and configured in IBM WebSphere MQ. It is accessible from a local queue manager. See WCF client to a .NET service hosted by WebSphere MQ sample and WCF client to an Axis Java service hosted by WebSphere MQ sample

Also use when you install WebSphere MQ web transport for SOAP. See Installing WebSphere MQ Web transport for SOAP

Message channel planning example for distributed platforms

This section provides a detailed example of how to connect two queue managers together so that messages can be sent between them.

The example illustrates the preparations required to enable an application using queue manager QM1 to put messages on a queue at queue manager QM2. An application running on QM2 can retrieve these messages, and send responses to a reply queue on QM1.

The example illustrates the use of TCP/IP connections. The example assumes that channels are to be triggered to start when the first message arrives on the transmission queue they are servicing. You must start the channel initiator in order for triggering to work.

This example uses SYSTEM.CHANNEL.INITQ as the initiation queue. This queue is already defined by WebSphere MQ. You can use a different initiation queue, but you must define it yourself and specify the name of the queue when you start the channel initiator.

What the example shows

The example shows the WebSphere MQ commands (MQSC) that you can use.

In all the examples, the MQSC commands are shown as they would appear in a file of commands, and as they would be typed at the command line. The two methods look identical, but, to issue a command at the command line, you must first type `runmqsc`, for the default queue manager, or `runmqsc` *qmname* where *qmname* is the name of the required queue manager. Then type any number of commands, as shown in the examples.

An alternative method is to create a file containing these commands. Any errors in the commands are then easy to correct. If you called your file mqsc.in then to run it on queue manager QMNAME use:

```
runmqsc QMNAME < mqsc.in > mqsc.out
```

You could verify the commands in your file before running it using:

```
runmqsc -v QMNAME < mqsc.in > mqsc.out
```

For portability, you should restrict the line length of your commands to 72 characters. Use a concatenation character to continue over more than one line. On Windows use Ctrl-z to end the input at the command line. On UNIX and Linux systems use Ctrl-d. Alternatively, use the **end** command.

Figure 4 shows the example scenario.

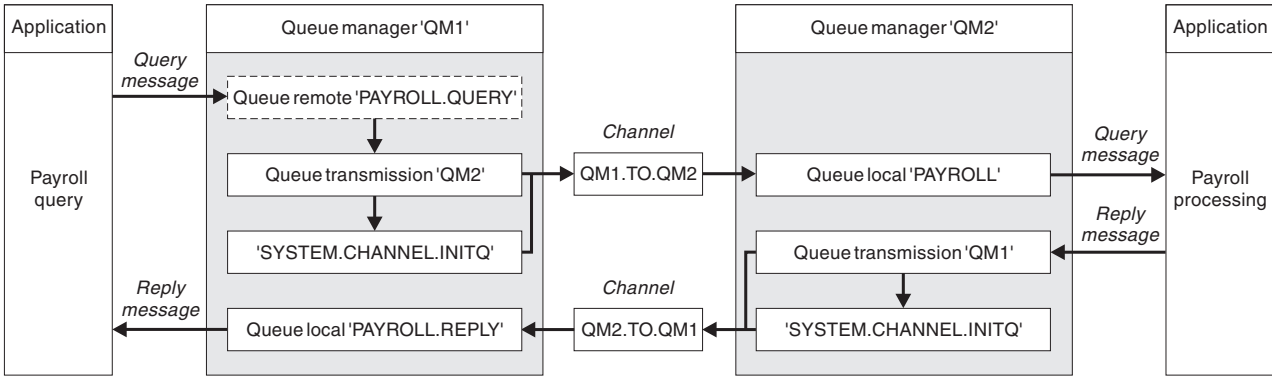


Figure 4. The message channel example for Windows, UNIX and Linux systems

The example involves a payroll query application connected to queue manager QM1 that sends payroll query messages to a payroll processing application running on queue manager QM2. The payroll query application needs the replies to its queries sent back to QM1. The payroll query messages are sent from QM1 to QM2 on a sender-receiver channel called QM1.TO.QM2, and the reply messages are sent back from QM2 to QM1 on another sender-receiver channel called QM2.TO.QM1. Both of these channels are triggered to start as soon as they have a message to send to the other queue manager.

The payroll query application puts a query message to the remote queue “PAYROLL.QUERY” defined on QM1. This remote queue definition resolves to the local queue “PAYROLL” on QM2. In addition, the payroll query application specifies that the reply to the query is sent to the local queue “PAYROLL.REPLY” on QM1. The payroll processing application gets messages from the local queue “PAYROLL” on QM2, and sends the replies to wherever they are required; in this case, local queue “PAYROLL.REPLY” on QM1.

In the example definitions for TCP/IP, QM1 has a host address of 192.0.2.0 and is listening on port 1411, and QM2 has a host address of 192.0.2.1 and is listening on port 1412. The example assumes that these are already defined on your system and available for use.

The object definitions that need to be created on QM1 are:

- Remote queue definition, PAYROLL.QUERY
- Transmission queue definition, QM2 (default=remote queue manager name)
- Sender channel definition, QM1.TO.QM2
- Receiver channel definition, QM2.TO.QM1
- Reply-to queue definition, PAYROLL.REPLY

The object definitions that need to be created on QM2 are:

- Local queue definition, PAYROLL
- Transmission queue definition, QM1 (default=remote queue manager name)
- Sender channel definition, QM2.TO.QM1
- Receiver channel definition, QM1.TO.QM2

The connection details are supplied in the CONNAME attribute of the sender channel definitions.

You can see a diagram of the arrangement in Figure 4 on page 125.

Queue manager QM1 example:

The following object definitions allow applications connected to queue manager QM1 to send request messages to a queue called PAYROLL on QM2, and to receive replies on a queue called PAYROLL.REPLY on QM1.

All the object definitions have been provided with the DESCR and REPLACE attributes. The other attributes supplied are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM1.

Run the following commands on queue manager QM1.

Remote queue definition

```
DEFINE QREMOTE(PAYROLL.QUERY) DESCR('Remote queue for QM2') REPLACE +
PUT(ENABLED) XMITQ(QM2) RNAME(PAYROLL) RQMNAME(QM2)
```

Note: The remote queue definition is not a physical queue, but a means of directing messages to the transmission queue, QM2, so that they can be sent to queue manager QM2.

Transmission queue definition

```
DEFINE QLOCAL(QM2) DESCR('Transmission queue to QM2') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
INITQ(SYSTEM.CHANNEL.INITQ) PROCESS(QM1.TO.QM2.PROCESS)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the named process.

Sender channel definition

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM2') XMITQ(QM2) +
CONNAME('192.0.2.1(1412)')
```

Receiver channel definition

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM2')
```

Reply-to queue definition

```
DEFINE QLOCAL(PAYROLL.REPLY) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Reply queue for replies to query messages sent to QM2')
```

The reply-to queue is defined as PUT(ENABLED). This ensures that reply messages can be put to the queue. If the replies cannot be put to the reply-to queue, they are sent to the dead-letter queue on QM1 or, if this queue is not available, remain on transmission queue QM1 on queue manager QM2. The queue has been defined as GET(ENABLED) to allow the reply messages to be retrieved.

Queue manager QM2 example:

The following object definitions allow applications connected to queue manager QM2 to retrieve request messages from a local queue called PAYROLL, and to put replies to these request messages to a queue called PAYROLL.REPLY on queue manager QM1.

You do not need to provide a remote queue definition to enable the replies to be returned to QM1. The message descriptor of the message retrieved from local queue PAYROLL contains both the reply-to queue and the reply-to queue manager names. Therefore, as long as QM2 can resolve the reply-to queue manager name to that of a transmission queue on queue manager QM2, the reply message can be sent. In this example, the reply-to queue manager name is QM1 and so queue manager QM2 requires a transmission queue of the same name.

All the object definitions have been provided with the DESCR and REPLACE attributes and are the minimum required to make the example work. The attributes that are not supplied take the default values for queue manager QM2.

Run the following commands on queue manager QM2.

Local queue definition

```
DEFINE QLOCAL(PAYROLL) REPLACE PUT(ENABLED) GET(ENABLED) +
DESCR('Local queue for QM1 payroll details')
```

This queue is defined as PUT(ENABLED) and GET(ENABLED) for the same reason as the reply-to queue definition on queue manager QM1.

Transmission queue definition

```
DEFINE QLOCAL(QM1) DESCR('Transmission queue to QM1') REPLACE +
USAGE(XMITQ) PUT(ENABLED) GET(ENABLED) TRIGGER TRIGTYPE(FIRST) +
INITQ(SYSTEM.CHANNEL.INITQ) PROCESS(QM2.TO.QM1.PROCESS)
```

When the first message is put on this transmission queue, a trigger message is sent to the initiation queue, SYSTEM.CHANNEL.INITQ. The channel initiator gets the message from the initiation queue and starts the channel identified in the named process.

Sender channel definition

```
DEFINE CHANNEL(QM2.TO.QM1) CHLTYPE(SDR) TRPTYPE(TCP) +
REPLACE DESCR('Sender channel to QM1') XMITQ(QM1) +
CONNAME('192.0.2.0(1411)')
```

Receiver channel definition

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) +
REPLACE DESCR('Receiver channel from QM1')
```

Running the example

Information about starting the channel initiator and listener and suggestions for expanding on this scenario.

Once these definitions have been created, you need to:

- Start the channel initiator on each queue manager.
- Start the listener for each queue manager.

For information about starting the channel initiator and listener, see Setting up communication for Windows and Setting up communication on UNIX and Linux systems.

Expanding this example

This simple example could be expanded with:

- The use of LU 6.2 communications for interconnection with CICS systems, and transaction processing.
- Adding more queue, process, and channel definitions to allow other applications to send messages between the two queue managers.
- Adding user-exit programs on the channels to allow for link encryption, security checking, or additional message processing.
- Using queue-manager aliases and reply-to queue aliases to understand more about how these can be used in the organization of your queue manager network.

Using an alias to refer to an MQ library

You can define an alias to refer to an MQ library in your JCL, rather than use the name of the MQ library directly. Then, if the name of the MQ library changes, you have only to delete and redefine the alias.

Example

The following example defines an alias MQM.SCSQANLE to refer to the MQ library MQM.V600.SCSQANLE:

```
//STEP1 EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN DD *
DEFINE ALIAS (NAME(MQM.SCSQANLE) RELATE(MQM.V600.SCSQANLE))
/*
```

Then, to refer to the MQM.V600.SCSQANLE library in your JCL, use the alias MQM.SCSQANLE.

Note: The library and alias names must be in the same catalog, so use the same high level qualifier for both; in this example, the high level qualifier is MQM.

Administration reference

Use the links to reference information in this section to help you operate and administer WebSphere MQ.

- Queue names
- Other object names
- "WebSphere MQ Administration Interface" on page 1254

Syntax diagrams

The syntax for a command and its options is presented in the form of a syntax diagram called a railroad diagram.

Railroad diagrams are a visual format suitable for sighted users; see, "How to read railroad diagrams" on page 129. It tells you what options you can supply with the command, how to enter them, indicates relationships between different options, and sometimes different values of an option.

How to read railroad diagrams

Each railroad diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a railroad diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in railroad diagrams are:

Table 33. How to read railroad diagrams

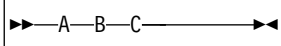
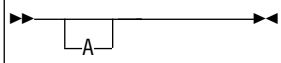
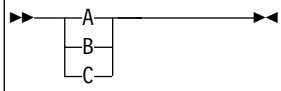
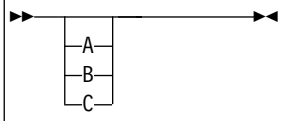
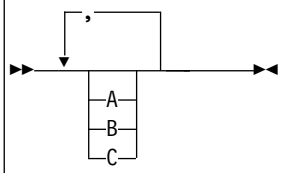
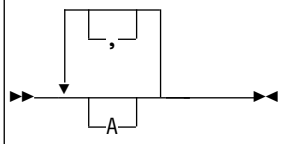
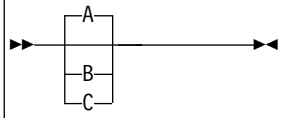
Convention	Meaning
	You must specify values A, B, and C. Required values are shown on the main line of a railroad diagram.
	You may specify value A. Optional values are shown below the main line of a railroad diagram.
	Values A, B, and C are alternatives, one of which you must specify.
	Values A, B, and C are alternatives, one of which you might specify.
	You might specify one or more of the values A, B, and C. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow.
	You might specify value A multiple times. The separator in this example is optional.
	Values A, B, and C are alternatives, one of which you might specify. If you specify none of the values shown, the default A (the value shown above the main line) is used.

Table 33. How to read railroad diagrams (continued)

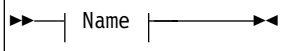
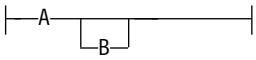
Convention	Meaning
 **Name:** 	The railroad fragment `Name` is shown separately from the main railroad diagram.

Table 33. How to read railroad diagrams (continued)

Convention	Meaning
Punctuation and uppercase values	Specify exactly as shown.

WebSphere MQ Control commands

Find out how to use the WebSphere MQ control commands.

If you want to issue control commands, your user ID must be a member of the mqm group. For more information, see Authority to administer WebSphere MQ on UNIX, Linux, and Windows systems .

When using control commands that operate on a queue manager, you must use the command from the installation associated with the queue manager you are working with.

In addition, note the following environment-specific information:

- On Windows, all control commands can be issued from a command line. Command names and their flags are not case-sensitive: you can enter them in uppercase, lowercase, or a combination of uppercase and lowercase. However, arguments to control commands (such as queue names) are case-sensitive.

  In the syntax descriptions, the hyphen (-) is used as a flag indicator. You can use the forward slash (/) instead of the hyphen.
- On UNIX and Linux systems, all WebSphere MQ control commands can be issued from a shell. All commands are case-sensitive.
- A subset of the control commands can be issued using the WebSphere MQ Explorer.

For a list of the control commands see, “The control commands” on page 133.

For a comparison of the different administration command sets, see “Comparing command sets” on page 252.

For information about commands for managing keys and certificates, see “Managing keys and certificates” on page 259.

Related concepts:

“MQSC reference” on page 276
Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

“Programmable command formats reference” on page 796
Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. PCFs simplify queue manager administration and other network administration.

Using control commands

The table in this topic shows the three categories of control commands: queue manager commands, channel commands, and utility commands.

Control commands can be divided into three categories, as shown in Table 34 on page 132.

Table 34. Categories of control commands

Category	Description
Queue manager commands	Queue manager control commands include commands for creating, starting, stopping, and deleting queue managers and command servers
Channel commands	Channel commands include commands for starting and ending channels and channel initiators
Utility commands	Utility commands include commands associated with: • Running MQSC commands • Conversion exits • Authority management • Recording and recovering media images of queue manager resources • Displaying and resolving transactions • Trigger monitors • Displaying the file names of WebSphere MQ objects

For more information, see "WebSphere MQ Control commands" on page 131

Using control commands on Windows systems:

In WebSphere MQ for Windows, you enter control commands at a command prompt.

In Windows environments, control commands and their flags are not case-sensitive, but arguments to those commands (such as queue names and queue-manager names) are case-sensitive.

For example, in the command:

```
crtmqm /u SYSTEM.DEAD.LETTER.QUEUE jupiter.queue.manager
```

- The command name can be entered in uppercase or lowercase, or a mixture of the two. These are all valid: `crtmqm`, `CRTMQM`, and `CRTmqm`.
- The flag can be entered as `-u`, `-U`, `/u`, or `/U`.
- `SYSTEM.DEAD.LETTER.QUEUE` and `jupiter.queue.manager` must be entered exactly as shown.

For more information, see WebSphere MQ control commands.

Using control commands on UNIX and Linux systems:

In WebSphere MQ for UNIX and Linux systems, you enter control commands in a shell window.

In UNIX environments, control commands, including the command name itself, the flags, and any arguments, are case-sensitive. For example, in the command:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE jupiter.queue.manager
```

- The command name must be `crtmqm`, not `CRTMQM`.
- The flag must be `-u`, not `-U`.
- The dead-letter queue is called `SYSTEM.DEAD.LETTER.QUEUE`.
- The argument is specified as `jupiter.queue.manager`, which is different from `JUPITER.queue.manager`.

Take care to type the commands exactly as you see them in the examples.

For more information about the **crtmqm** command, see "**crtmqm**" on page 146.

For more information on control commands, see "WebSphere MQ Control commands" on page 131

The control commands

This collection of topics provides reference information for each of the WebSphere MQ control commands. These control commands require that the ID is in the mqm group.

addmqinf:

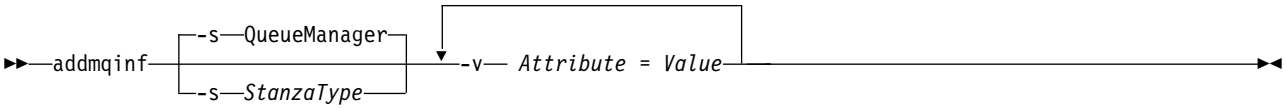
Add WebSphere MQ configuration information (Windows and UNIX platforms only).

Purpose

Use the **addmqinf** command to add information to the WebSphere MQ configuration data.

For example, use **dspmqinf** and **addmqinf** to copy configuration data from the system where a queue manager was created, to other systems where the same multi-instance queue manager is also to be started.

Syntax



Required parameters

-v *Attribute=Value*
The name and value of the stanza attributes to be placed in the stanza specified in the command.

Table 35 lists the QueueManager stanza attribute values. The queue manager stanza is the only stanza that is currently supported.

Table 35. QueueManager stanza attributes

Attribute	*Value*	**Required or optional**
Name	The name of the queue manager. You must provide a different name from any other queue manager stanza on the system.	Required
Prefix	The directory path *under* which this queue manager data directory is stored by default. You can use **Prefix** to modify the location of the queue manager data directories. The value of **Directory** *is* automatically appended to this path.	Required
Directory	The name of the queue manager data directory. Sometimes the name must be provided (as in "Example" on page 134), because it is different from the queue manager name. Copy the directory name from the value returned by **dspmqinf**. The rules for transforming queue manager names into directory names are described in Understanding WebSphere MQ file names.	Required
DataPath	The directory path where the queue manager data files are placed. The value of **Directory** is *not* automatically appended to this path - you must provide the transformed queue manager name as part of **DataPath**. If the **DataPath** attribute is omitted on UNIX, the queue manager data directory path is defined as ***Prefix*/*Directory***.	UNIX: Optional Windows: Required

Optional parameters

-s *StanzaType*
A stanza of the type *StanzaType* is added to the WebSphere MQ configuration.

The default value of *StanzaType* is `QueueManager`.

The only supported value of *StanzaType* is `QueueManager`.

Return codes

Return code	Description
0	Successful operation
1	Queue manager location is invalid (either **Prefix** or **DataPath**)
39	Bad command-line parameters
45	Stanza already exists
46	Required configuration attribute is missing
58	Inconsistent use of installations detected
69	Storage is not available
71	Unexpected error
72	Queue manager name error
100	Log location is invalid

Example

```
addmqinf -v DataPath=/MQHA/qmgrs/QM!NAME  +
  -v Prefix=/var/mqm  +
  -v Directory=QM!NAME  +
  -v Name=QM.NAME
```

Creates the following stanza in mqs.ini:

```
QueueManager:
   Name=QM.NAME
   Prefix=/var/mqm
   Directory=QM!NAME
   DataPath=/MQHA/qmgrs/QM!NAME
```

Usage notes

Use **dspmqinf** with **addmqinf** to create an instance of a multi-instance queue manager on a different server.

To use this command you must be a WebSphere MQ administrator and a member of the `mqm` group.

Related commands

Command	Description
"dspmqinf" on page 172	Display WebSphere MQ configuration information
"rmvmqinf" on page 204	Remove WebSphere MQ configuration information

amqmdain:

amqmdain is used to configure or control some Windows specific administrative tasks.

Purpose

The **amqmdain** command applies to IBM WebSphere MQ for Windows only.

Use **amqmdain** to perform some Windows specific administrative tasks.

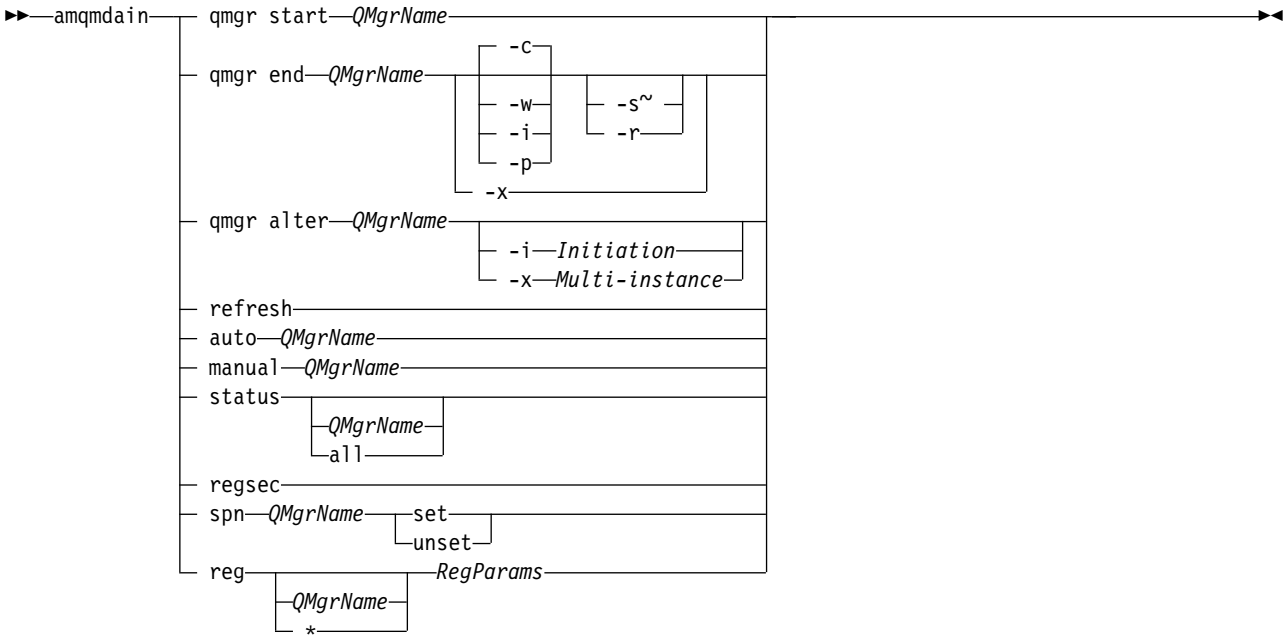
Starting a queue manager with **amqmdain** is equivalent to using the **strmqm** command with the option `-ss`. **amqmdain** makes the queue manager run in a non-interactive session under a different user account. However, to ensure that all queue manager startup feedback is returned to the command line, use the `strmqm -ss` command rather than **amqmdain**.

You must use the **amqmdain** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

To administer and define IBM WebSphere MQ service and listener objects, use MQSC commands, PCF commands, or the IBM WebSphere MQ Explorer.

The **amqmdain** command has been updated to modify either the `.ini` files or the registry as appropriate.

Syntax



Keywords and parameters

All parameters are required unless the description states they are optional.

In every case, *QMgrName* is the name of the queue manager to which the command applies.

qmgr start *QMgrName*
Starts a queue manager.

This parameter can also be written in the form *start QMgrName*.

If you start your queue manager as a service and need the queue manager to continue to run after logoff, use `strmqm -ss qmgr` instead of `amqmdain start qmgr`.

qmgr end *QMgrName*
Ends a queue manager.

This parameter can also be written in the form *end QMgrName*.

For consistency across platforms, use `endmqm qmgr` instead of `amqmdain end qmgr`.

For fuller descriptions of the options, see "endmqm" on page 189.

-c Controlled (or quiesced) shutdown.

-w Wait shutdown.

-i Immediate shut down.

-p Pre-emptive shut down.

-r Reconnect clients.

-s Switch over to a standby queue manager instance.

-x End the standby instance of the queue manager without ending the active instance.

qmgr alter *QMgrName*
Alters a queue manager.

-i *Initiation*
Specifies the initiation type. Possible values are:

Table 36. Initiation parameters.

Parameter	Parameter Description
auto	Sets the queue manager to automatic startup (when the machine starts, or more precisely when the IBM WebSphere MQ service starts). The syntax is: `amqmdain qmgr alter QmgrName -i auto`
interactive	Sets the queue manager to manual startup that then runs under the logged on (interactive) user. The syntax is: `amqmdain qmgr alter QmgrName -i interactive`
service	Sets the queue manager to manual startup that then runs as a service. The syntax is: `amqmdain qmgr alter QmgrName -i service`

-x *Multi-instance*
Specifies if **auto** queue manager start by the IBM WebSphere MQ service permits multiple instances. Equivalent to the `-sax` option on the **crtmqm** command. Also specifies if the **amqmdain start qmgr** command permits standby instances. Possible values are:

Table 37. Multi-instance parameters.

Parameter	Parameter Description
set	Sets automatic queue manager startup to permit multiple instances. Issues `strmqm -x`. The `set` option is ignored for queue managers that are initiated interactively or as a manual service startup. The syntax of the command is: `amqmdain qmgr alter QmgrName -x set`
unset	Sets automatic queue manager startup to single instance. Issues `strmqm`. The `unset` option is ignored for queue managers that are initiated interactively or as a manual service startup. The syntax of the command is: `amqmdain qmgr alter QmgrName -x unset`

refresh
Refreshes or checks the status of a queue manager. You will not see anything returned on the screen after executing this command.

auto *QMgrName*
Sets a queue manager to automatic startup.

manual *QMgrName*
Sets a queue manager to manual startup.

status *QMgrName* | **all**
These parameters are optional.

Table 38. Status parameters.

Parameter	Parameter Description
If no parameter is supplied:	Displays the status of the IBM WebSphere MQ services.
If a *QMgrName* is supplied:	Displays the status of the named queue manager.
If the parameter *all* is supplied:	Displays the status of the IBM WebSphere MQ services and all queue managers.

regsec
Ensures that the security permissions assigned to the Registry keys containing installation information are correct.

spn *QMgrName* **set** | **unset**
You can set or unset the service principal name for a queue manager.

reg *QMgrName* | * *RegParams*
Parameters *QMgrName*, and * are optional.

Table 39. Reg parameters.

Parameter	Parameter Description
If *RegParams* is specified alone:	Modifies **queue manager** configuration information related to the default queue manager.
If *QMgrName* and *RegParams* are specified:	Modifies **queue manager** configuration information related to the queue manager specified by *QMgrName*.
If * and *RegParams* are specified:	Modifies **IBM WebSphere MQ** configuration information.

The parameter, *RegParams*, specifies the stanzas to change, and the changes that are to be made. *RegParams* takes one of the following forms:

- `-c add -s stanza -v attribute=value`
- `-c remove -s stanza -v [attribute|*]`
- `-c display -s stanza -v [attribute|*]`

If you are specifying queue manager configuration information, the valid values for *stanza* are:

```
XAResourceManager\name
ApiExitLocal\name
Channels
ExitPath
InstanceData
Log
QueueManagerStartup
TCP
LU62
SPX
NetBios
Connection
```

```
QMErrorLog
Broker

ExitPropertiesLocal
SSL
```

If you are modifying IBM WebSphere MQ configuration information, the valid values for *stanza* are:

```
ApiExitCommon\name
ApiExitTemplate\name
ACPI
AllQueueManagers
Channels
DefaultQueueManager
LogDefaults
ExitProperties
```

The following are usage considerations:

- **amqmdain** does not validate the values you specify for *name*, *attribute*, or *value*.
- When you specify add, and an attribute exists, it is modified.
- If a stanza does not exist, **amqmdain** creates it.
- When you specify remove, you can use the value * to remove all attributes.
- When you specify display, you can use the value * to display all attributes which have been defined. This value only displays the attributes which have been defined and not the complete list of valid attributes.
- If you use remove to delete the only attribute in a stanza, the stanza itself is deleted.
- Any modification you make to the Registry re-secures all IBM WebSphere MQ Registry entries.

Examples

The following example adds an XAResourceManager to queue manager TEST. The commands issued are:

```
amqmdain reg TEST -c add -s XAResourceManager\Sample -v SwitchFile=sf1
amqmdain reg TEST -c add -s XAResourceManager\Sample -v ThreadOfControl=THREAD
amqmdain reg TEST -c add -s XAResourceManager\Sample -v XAOpenString=openit
amqmdain reg TEST -c add -s XAResourceManager\Sample -v XACloseString=closeit
```

To display the values set by the commands above, use:

```
amqmdain reg TEST -c display -s XAResourceManager\Sample -v *
```

The display would look something like the following:

```
0784726, 5639-B43 (C) Copyright IBM Corp. 1994, 2002. ALL RIGHTS RESERVED.
Displaying registry value for Queue Manager 'TEST'
        Attribute = Name, Value = Sample
        Attribute = SwitchFile, Value = sf1
        Attribute = ThreadOfControl, Value = THREAD
        Attribute = XAOpenString, Value = openit
        Attribute = XACloseString, Value = closeit
```

To remove the XAResourceManager from queue manager TEST, use:

```
amqmdain reg TEST -c remove -s XAResourceManager\Sample -v *
```

Return codes

Return code	Description
0	Command completed normally
-2	Syntax error
-3	Failed to initialize MFC
-6	Feature no longer supported
-7	Configuration failed
-9	Unexpected Registry error
-16	Failed to configure service principal name
-29	Inconsistent use of installations detected
62	The queue manager is associated with a different installation
71	Unexpected error
119	Permission denied (Windows only)

Note:

1. If the *qmgr start QMgrName* command is issued, all return codes that can be returned with **strmqm**, can be returned here also. For a list of these return codes, see "strmqm" on page 243.
2. If the *qmgr end QMgrName* command is issued, all return codes that can be returned with **endmqm**, can be returned here also. For a list of these return codes, see "endmqm" on page 189.

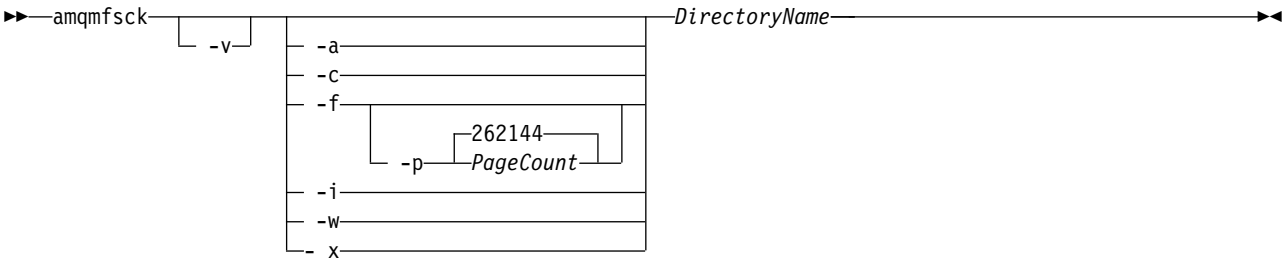
amqmfsck (file system check):

amqmfsck checks whether a shared file system on UNIX and IBM i systems meets the requirements for storing the queue manager data of a multi-instance queue manager.

Purpose

The **amqmfsck** command applies only to UNIX and IBM i systems. You do not need to check the network drive on Windows. **amqmfsck** tests that a file system correctly handles concurrent writes to a file and the waiting for and releasing of locks.

Syntax



Required parameters

DirectoryName
The name of the directory to check.

Optional parameters

-a Perform the second phase of the data integrity test.

Run this on two machines at the same time. You must have formatted the test file using the **-f** option previously

-c Test writing to a file in the directory concurrently.

-f Perform the first phase of the data integrity test.

Formats a file in the directory in preparation for data integrity testing.

-i Perform the third phase of the data integrity test.

Checks the integrity of the file after the failure to discover whether the test worked.

-p Specifies the size of the test file used in the data integrity test in pages. .

The size is rounded up to the nearest multiple of 16 pages. The file is formatted with *PageCount* pages of 4 KB.

The optimum size of the file depends on the speed of the filesystem and the nature of the test you perform. If this parameter is omitted, the test file is 262144 pages, or 1 GB.

The size is automatically reduced so that the formatting completes in about 60 seconds even on a very slow filesystem.

-v Verbose output.

-w Test waiting for and releasing locks.

-x Deletes any files created by **amqmfsck** during the testing of the directory.

Do not use this option until you have completed the testing, or if you need to change the number of pages used in the integrity test.

Usage

You must be a WebSphere MQ Administrator to run the command. You must have read/write access to the directory being checked.

The command returns an exit code of zero if the tests complete successfully.

The task, Verifying shared file system behavior, describes how to use **amqmfsck** to check the whether of a file system is suitable for multi-instance queue managers.

Interpreting your results

If the check fails, the file system is not capable of being used by WebSphere MQ queue managers. If the tests fail, choose verbose mode to help you to interpret the errors. The output from the `verbose` option helps you understand why the command failed, and if the problem can be solved by reconfiguring the file system.

Sometimes the failure might be an access control problem that can be fixed by changing directory ownership or permissions. Sometimes the failure can be fixed by reconfiguring the file system to behave in a different way. For example, some file systems have performance options that might need to be changed. It is also possible that the file system protocol does not support concurrency sufficiently robustly, and you must use a different file system. For example, you must use NFSv4 rather than NFSv3.

If the check succeeds, the command reports `The tests on the directory completed successfully`. If your environment is not listed as supported in the testing and support statement, this result does not necessarily mean that you can run IBM WebSphere MQ multi-instance queue managers successfully. You must plan and run a variety of tests to satisfy yourself that you have covered all foreseeable circumstances. Some failures are intermittent, and there is a better chance of discovering them if you run the tests more than once.

Related information:
Verifying shared file system behavior

crtmqcvx:

Create data conversion code from data type structures.

Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert C structures.

The command reads an input file containing structures to be converted, and writes an output file containing code fragments to convert those structures.

For information about using this command, see Utility for creating conversion-exit code.

Syntax

Required parameters

SourceFile
The input file containing the C structures to convert.

TargetFile
The output file containing the code fragments generated to convert the structures.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following example shows the results of using the data conversion command against a source C structure. The command issued is:

```
crtmqcvx source.tmp target.c
```

The input file, `source.tmp`, looks like this:

```
/* This is a test C structure which can be converted by the */
/* crtmqcvx utility                                         */

struct my_structure
{
    int    code;
    MQLONG value;
};
```

The output file, `target.c`, produced by the command, looks like this:

```
MQLONG Convertmy_structure(
              PMQDXP  pExitParms,
              PMQBYTE *in_cursor,
              PMQBYTE *out_cursor,
              PMQBYTE in_lastbyte,
              PMQBYTE out_lastbyte,
              MQHCONN hConn,
              MQLONG  opts,
              MQLONG  MsgEncoding,
              MQLONG  ReqEncoding,
              MQLONG  MsgCCSID,
              MQLONG  ReqCCSID,
              MQLONG  CompCode,
              MQLONG  Reason)
{
    MQLONG ReturnCode = MQRC_NONE;

    ConvertLong(1); /* code */

    AlignLong();
    ConvertLong(1); /* value */

Fail:
    return(ReturnCode);
}
```

You can use these code fragments in your applications to convert data structures. However, if you do so, the fragment uses macros supplied in the header file `amqsvmha.h`.

crtmqenv:

Create a list of environment variables for an installation of IBM WebSphere MQ, on UNIX, Linux, and Windows.

Purpose

You can use the **crtmqenv** command to create a list of environment variables with the appropriate values for an installation of IBM WebSphere MQ. The list of environment variables is displayed on the command line, and any variables that exist on the system have the IBM WebSphere MQ values added to them. This command does not set the environment variables for you, but gives you the appropriate strings to set the variables yourself, for example, within your own scripts.

If you want the environment variables set for you in a shell environment, you can use the **setmqenv** command instead of using the **crtmqenv** command.

You can specify which installation the environment is created for by specifying a queue manager name, an installation name, or an installation path. You can also create the environment for the installation that issues the **crtmqenv** command by issuing the command with the **-s** parameter.

This command lists the following environment variables, and their values, appropriate to your system:

- CLASSPATH
- INCLUDE
- LIB
- MANPATH
- MQ_DATA_PATH
- MQ_ENV_MODE

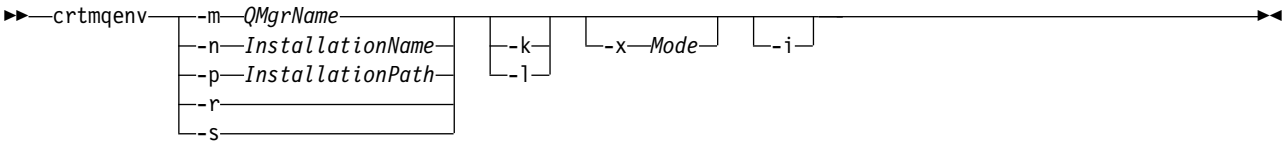
- MQ_FILE_PATH
- MQ_JAVA_INSTALL_PATH
- MQ_JAVA_DATA_PATH
- MQ_JAVA_LIB_PATH
- MQ_JAVA_JVM_FLAG
- MQ_JRE_PATH
- PATH

On UNIX and Linux systems, if the **-l** or **-k** flag is specified, the *LIBPATH* environment variable is set on AIX, and the *LD_LIBRARY_PATH* environment variable is set on HP-UX, Linux, and Solaris.

Usage notes

The **crtmqenv** command removes all directories for all IBM WebSphere MQ installations from the environment variables before adding new references to the installation for which you are setting up the environment. Therefore, if you want to set any additional environment variables that reference IBM WebSphere MQ, set the variables after issuing the **crtmqenv** command. For example, if you want to add *MQ_INSTALLATION_PATH*/java/lib to *LD_LIBRARY_PATH*, you must do so after running **crtmqenv**.

Syntax



Required Parameters

-m *QMgrName*
Create the environment for the installation associated with the queue manager *QMgrName*.

-n *InstallationName*
Create the environment for the installation named *InstallationName*.

-p *InstallationPath*
Create the environment for the installation in the path *InstallationPath*.

-r Remove all installations from the environment.

-s Create the environment for the installation that issued the command.

Optional Parameters

-k UNIX and Linux only.

Include the *LD_LIBRARY_PATH* , or *LIBPATH*, environment variable in the environment, adding the path to the IBM WebSphere MQ libraries at the start of the current *LD_LIBRARY_PATH*, or *LIBPATH*, variable.

-l UNIX and Linux only.

Include the *LD_LIBRARY_PATH*, or *LIBPATH*, environment variable in the environment, adding the path to the IBM WebSphere MQ libraries at the end of the current *LD_LIBRARY_PATH*, or *LIBPATH*, variable.

-x *Mode*
Mode can take the value 32, or 64.

Create a 32-bit or 64-bit environment. If this parameter is not specified, the environment matches that of the queue manager or installation specified in the command.

Any attempt to display a 64-bit environment with a 32-bit installation fails.

-i List only the additions to the environment.

When this parameter is specified, the environment variables set for previous installations remain in the environment variable path and must be manually removed.

Return codes

Return code	Description
0	Command completed normally.
10	Command completed with unexpected results.
20	An error occurred during processing.

Examples

The following examples assume that a copy of IBM WebSphere MQ is installed in `/opt/mqm` on a UNIX or Linux system.

1. This command creates a list of environment variables for an installation installed in `/opt/mqm`:
   ```
   /opt/mqm/bin/crtmqenv -s
   ```
2. This command creates a list of environment variables for an installation installed in `/opt/mqm2`, and includes the path to the installation at the end of the current value of the *LD_LIBRARY_PATH* variable:
   ```
   /opt/mqm/bin/crtmqenv -p /opt/mqm2 -l
   ```
3. This command creates a list of environment variables for the queue manager `QM1`, in a 32-bit environment:
   ```
   /opt/mqm/bin/crtmqenv -m QM1 -x 32
   ```

The following example assumes that a copy of IBM WebSphere MQ is installed in `c:\Program Files\IBM\WebSphere MQ` on a Windows system.

1. This command creates a list of environment variables for an installation called `installation1`:
   ```
   "c:\Program Files\IBM\WebSphere MQ\crtmqenv" -n installation1
   ```

Related reference:

"setmqenv" on page 231
Use the **setmqenv** to set up the IBM WebSphere MQ environment, on UNIX, Linux, and Windows.

Related information:

Choosing a primary installation

Multiple installations

crtmqinst:

Create installation entries in `mqinst.ini` on UNIX and Linux systems.

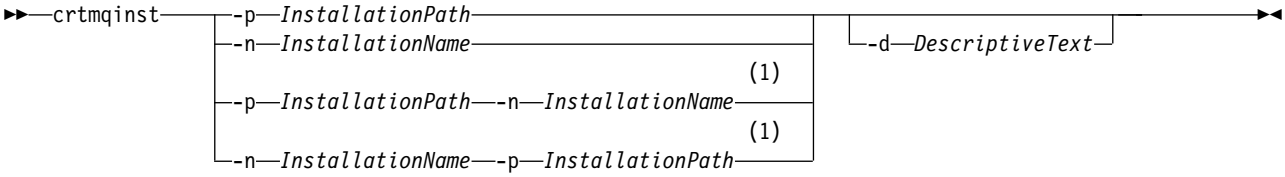
Purpose

File `mqinst.ini` contains information about all IBM WebSphere MQ installations on a system. For more information about mqinst.ini, see Installation configuration file, mqinst.ini .

The first IBM WebSphere MQ installation is automatically given an installation name of `Installation1` because the **crtmqinst** command is not available until an installation of IBM WebSphere MQ is on the system. Subsequent installations can have an installation name set before installation occurs, by using the

crtmqinst command. The installation name cannot be changed after installation. For more information about installation names, see Choosing an installation name.

Syntax



Notes:

1 When specified together, the installation name and installation path must refer to the same installation.

Parameters

-d Text that describes the installation.

The text can be up to 64 single-byte characters, or 32 double-byte characters. The default value is all blanks. You must use quotation marks around the text if it contains spaces.

-n *InstallationName*
The name of the installation.

The name can contain up to 16 single-byte characters and must be a combination of alphabetic and numeric characters in the ranges a-z, A-Z, and 0-9. The installation name must be unique, regardless of whether uppercase or lowercase characters are used. For example, the names `INSTALLATIONNAME` and `InstallationName` are not unique. If you do not supply the installation name, the next available name in the series Installation1, Installation2... is used.

-p *InstallationPath*
The installation path. If you do not supply the installation path, `/opt/mqm` is used on UNIX and Linux systems, and `/usr/mqm` is used on AIX.

Return codes

Return code	Description
0	Entry created without error
10	Invalid installation level
36	Invalid arguments supplied
37	Descriptive text was in error
45	Entry already exists
59	Invalid installation specified
71	Unexpected error
89	.ini file error
96	Could not lock .ini file
98	Insufficient authority to access .ini file
131	Resource problem

Example

1. This command creates an entry with an installation name of `myInstallation`, an installation path of `/opt/myInstallation`, and a description “My WebSphere MQ installation”:

```
crtmqinst -n MyInstallation -p /opt/myInstallation -d "My WebSphere MQ installation"
```

Quotation marks are needed because the descriptive text contains spaces.

Note: On UNIX systems, the **crtmqinst** command must be run by the root user because full access permissions are required to write to the mqinst.ini configuration file.

crtmqm:

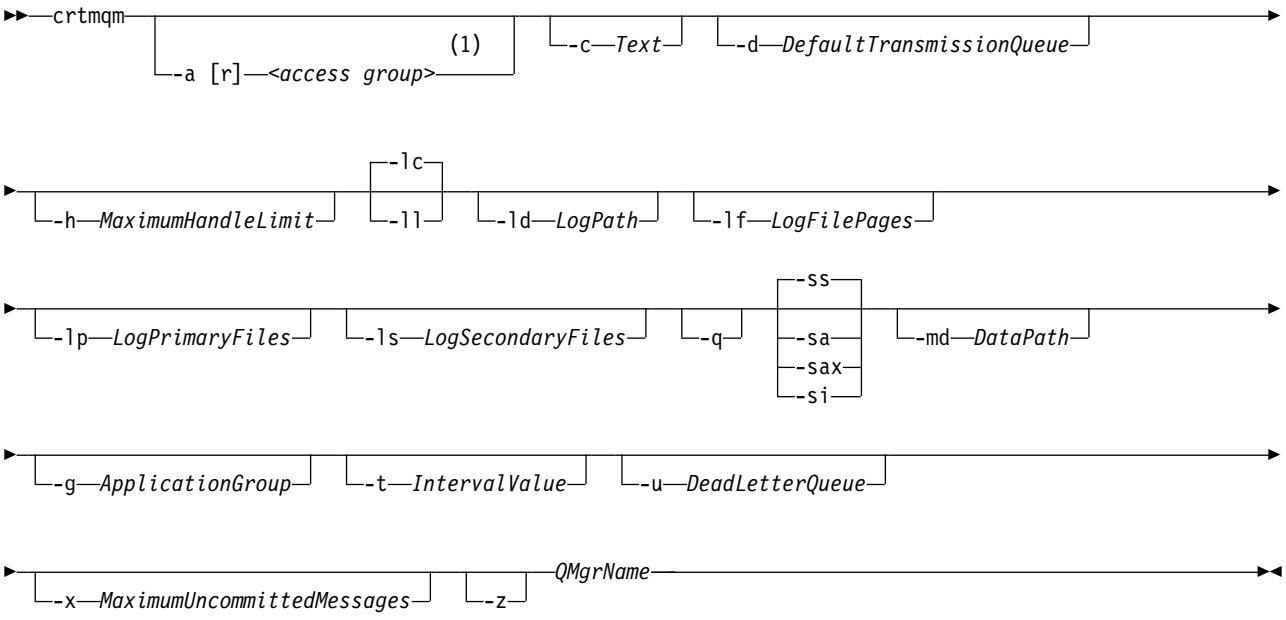
Create a queue manager.

Purpose

Use the **crtmqm** command to create a queue manager and define the default and system objects. The objects created by the **crtmqm** command are listed in System and default objects. When you have created a queue manager, use the **strmqm** command to start it.

The queue manager is automatically associated with the installation from which the **crtmqm** command was issued. To change the associated installation, use the **setmqm** command. Note that the Windows installer does not automatically add the user that performs the installation to the mqm group. For more details, see Authority to administer WebSphere MQ on UNIX, Linux and Windows systems .

Syntax



Notes:

1 Windows only

Required parameters

QMgrName
The name of the queue manager that you want to create. The name can contain up to 48 characters. This parameter must be the last item in the command.

Note: WebSphere MQ checks if the queue manager name exists. If the name already exists in the directory, then a suffix of .000,.001,.002, and so on, is added to the queue manager name. For example, if a queue manager QM1 is added to the directory and if QM1 already exists, then a queue manager with the name QM1.000 (suffix .000) is created.

Optional parameters

-a[r]*access group*

Use the access group parameter to specify a Windows security group, members of which will be granted full access to all queue manager data files. The group can either be a local or global group, depending on the syntax used.

Valid syntax for the group name is as follows:

LocalGroup

Domain name \ GlobalGroup name

GlobalGroup name@Domain name

You must define the additional access group before running the **crtmqm** command with the **-a [r]** option.

If you specify the group using **-ar** instead of **--a** , the local mqm group is not granted access to the queue manager data files. Use this option if the file system hosting the queue manager data files does not support access control entries for locally defined groups.

The group is typically a global security group, which is used to provide multi-instance queue managers with access to a shared queue manager data and logs folder. Use the additional security access group to set read and write permissions on the folder or to share containing queue manager data and log files.

The additional security access group is an alternative to using the local group named mqm to set permissions on the folder containing queue manager data and logs. Unlike the local group mqm, you can make the additional security access group a local or a global group. It must be a global group to set permissions on the shared folders that contain the data and log files used by multi-instance queue managers.

The Windows operating system checks the access permissions to read and write queue manager data and log files. It checks the permissions of the user ID that is running queue manager processes. The user ID that is checked depends on whether you started the queue manager as a service or you started it interactively. If you started the queue manager as a service, the user ID checked by the Windows system is the user ID you configured with the Prepare IBM WebSphere MQ wizard. If you started the queue manager interactively, the user ID checked by the Windows system is the user ID that ran the **strmqm** command.

The user ID must be a member of the local mqm group to start the queue manager. If the user ID is a member of the additional security access group, the queue manager can read and write files that are given permissions by using the group.

Restriction: You can specify an additional security access group only on Windows operating system. If you specify an additional security access group on other operating systems, the **crtmqm** command returns an error.

-c *Text*

Descriptive text for this queue manager. You can use up to 64 characters; the default is all blanks.

If you include special characters, enclose the description in single quotation marks. The maximum number of characters is reduced if the system is using a double-byte character set (DBCS).

-d *DefaultTransmissionQueue*

The name of the local transmission queue where remote messages are put if a transmission queue is not explicitly defined for their destination. There is no default.

-g *ApplicationGroup*

The name of the group that contains members that are allowed to perform the following actions:

- Run MQI applications
- Update all IPCC resources

- Change the contents of some queue manager directories

This option applies to IBM WebSphere MQ for AIX, Solaris, HP-UX, and Linux.

The default value is **-g all** , which allows unrestricted access.

The **-g** *ApplicationGroup* value is recorded in the queue manager configuration file named, `qm.ini`.

The `mqm` user ID and the user running the command must belong to the specified Application Group. For further details of the operation of restricted mode, see Restricted mode.

-h *MaximumHandleLimit*
The maximum number of handles that an application can open at the same time.

Specify a value in the range 1 - 999999999. The default value is 256.

The next set of parameter descriptions relate to logging, which is described in Using the log for recovery.

Note: Choose the logging arrangements with care, because some cannot be changed after they are committed.

-lc
Use circular logging. This method is the default logging method.

-ld *LogPath*
The directory used to store log files. The default directory to store log paths is defined when you install IBM WebSphere MQ.

If the volume containing the log file directory supports file security, the log file directory must have access permissions. The permissions allow the user IDs, under whose authority the queue manager runs, read and write access to the directory and its subdirectories. When you install IBM WebSphere MQ, you grant permissions to the user IDs and to the `mqm` group on the default log directory. If you set the *LogPath* parameter to write the log file to a different directory, you must grant the user IDs permission to read and write to the directory. The user ID and permissions for UNIX and Linux are different from those for the Windows system:

UNIX and Linux
The directory and its subdirectories must be owned by the user `mqm` in the group `mqm`.

If the log file is shared between different instances of the queue manager, the security identifiers (sid) that are used must be the same for the different instances. You must have set the user `mqm` to the same `sid` on the different servers running instances of the queue manager. Likewise for the group `mqm` .

Windows
If the directory is accessed by only one instance of the queue manager, you must give read and write access permission to the directory for the following groups and users:

- The local group `mqm`
- The local group `Administrators`
- The `SYSTEM` user ID

To give different instances of a queue manager access to the shared log directory, the queue manager must access the log directory using a global user. Give the global group, which contains the global user, read and write access permission to the log directory. The global group is the additional security access group specified in the **-a** parameter.

In IBM WebSphere MQ for Windows systems, the default directory is `C:\Program Files\IBM\WebSphere MQ\log` (assuming that **C** is your data drive). If the volume supports file security, the SYSTEM ID, Administrators, and `mqm` group must be granted read/write access to the directory.

In IBM WebSphere MQ for UNIX and Linux systems, the default directory is `/var/mqm/log`. User ID `mqm` and group `mqm` must have full authorities to the log files.

If you change the locations of these files, you must give these authorities yourself. If these authorities are set automatically, then the log files are in their default locations.

-lf *LogFilePages*
The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

In IBM WebSphere MQ for UNIX and Linux systems, the default number of log file pages is 4096, giving a log file size of 16 MB. The minimum number of log file pages is 64 and the maximum is 65535.

In IBM WebSphere MQ for Windows systems, the default number of log file pages is 4096, giving a log file size of 16 MB. The minimum number of log file pages is 32 and the maximum is 65535.

Note: The size of the log files for a queue manager specified during creation of that queue manager cannot be changed.

-ll*LinearLogging*
Use linear logging.

-lp *LogPrimaryFiles*
The log files allocated when the queue manager is created.

On a Windows system, the minimum number of primary log files you can have is 2 and the maximum is 254. On UNIX and Linux systems, the minimum number of primary log files you can have is 2 and the maximum is 510. The default is 3.

On a Windows system, the total number of primary and secondary log files must not exceed 255 and must not be less than 3. On UNIX and Linux systems the total number of primary and secondary log files must not exceed 511 and must not be less than 3.

Operating system limits can reduce the maximum log size.

The value is examined when the queue manager is created or started. You can change it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted, and the effect might not be immediate.

For more information about primary log files, see What logs look like .

To calculate the size of the primary log files, see Calculating the size of the log .

-ls *LogSecondaryFiles*
The log files allocated when the primary files are exhausted.

On a Windows system, the minimum number of secondary log files you can have is 1 and the maximum is 253. On UNIX and Linux systems, the minimum number of secondary log files you can have is 2 and the maximum is 509. The default is 2.

On a Windows system, the total number of secondary and secondary log files must not exceed 255 and must not be less than 3. On UNIX and Linux systems the total number of primary and secondary log files must not exceed 511 and must not be less than 3.

Operating system limits can reduce the maximum log size.

The value is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect might not be immediate.

For more information about the use of secondary log files, see What logs look like.

To calculate the size of the secondary log files, see Calculating the size of the log .

-md *DataPath*
The directory used to hold the data files for a queue manager.

In IBM WebSphere MQ for Windows systems, the default is `C:\Program Files\IBM\WebSphere MQ\qmgrs` (assuming that **C:** is your data drive). If the volume supports file security, the SYSTEM ID, Administrators, and `mqm` group must be granted read/write access to the directory.

In IBM WebSphere MQ for UNIX and Linux systems, the default is `/var/mqm/qmgrs`. User ID `mqm` and group `mqm` must have full authorities to the log files.

The `DataPath` parameter is provided to assist in the configuration of multi-instance queue managers. For example, on UNIX and Linux systems: if the `/var/mqm` directory is located on a local file system, use the `DataPath` parameter and the `LogPath` parameter to point to the shared file systems accessible to multiple queue managers.

Note: A queue manager created using `DataPath` parameter runs on versions of WebSphere MQ earlier than version 7.0.1, but the queue manager must be reconfigured to remove the `DataPath` parameter. You have two options to restore the queue manager to a pre-version 7.0.1 configuration and run without the `DataPath` parameter: If you are confident about editing queue manager configurations, you can manually configure the queue manager using the `Prefix` queue manager configuration parameter. Alternatively, complete the following steps to edit the queue manager:

1. Stop the queue manager.
2. Save the queue manager data and log directories.
3. Delete the queue manager.
4. Backout WebSphere MQ to the pre-v7.0.1 fix level.
5. Create the queue manager with the same name.
6. Replace the new queue manager data and log directories with the ones you saved.

-q Makes this queue manager the default queue manager. The new queue manager replaces any existing default queue manager.

If you accidentally use this flag and you want to revert to an existing queue manager as the default queue manager, change the default queue manager as described in Making an existing queue manager the default.

-sa

Automatic queue manager startup. For Windows systems only.

The queue manager is configured to start automatically when the IBM WebSphere MQ Service starts.

This is the default option if you create a queue manager from IBM WebSphere MQ Explorer.

Queue managers created in IBM WebSphere MQ releases earlier than Version 7 retain their existing startup type.

-sax

Automatic queue manager startup, permitting multiple instances. For Windows systems only.

The queue manager is configured to start automatically when the IBM WebSphere MQ Service starts.

If an instance of the queue manager is not already running the queue manager starts, the instance becomes active, and standby instances are permitted elsewhere. If a queue manager instance that permits standbys is already active on a different server, the new instance becomes a standby instance.

Only one instance of a queue manager can run on a server.

Queue managers created in IBM WebSphere MQ versions earlier than Version 7.0.1 retain their existing startup type.

-si

Interactive (manual) queue manager startup.

The queue manager is configured to start only when you manually request startup by using the **strmqm** command. The queue manager runs under the (interactive) user when that user is logged-on. Queue managers configured with interactive startup end when the user who started them logs off.

-ss
Service (manual) queue manager startup.

A queue manager configured to start only when manually requested by using the **strmqm** command. The queue manager then runs as a child process of the service when the IBM WebSphere MQ Service starts. Queue managers configured with service startup continue to run even after the interactive user has logged off.

This is the default option if you create a queue manager from the command line.

-t *IntervalValue*
The trigger time interval in milliseconds for all queues controlled by this queue manager. This value specifies the length of time triggering is suspended, after the queue manager receives a trigger-generating message. That is, if the arrival of a message on a queue causes a trigger message to be put on the initiation queue, any message arriving on the same queue within the specified interval does not generate another trigger message.

You can use the trigger time interval to ensure that your application is allowed sufficient time to deal with a trigger condition before it is alerted to deal with another trigger condition on the same queue. You might choose to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range 0 - 999999999. The default is 999999999 milliseconds; a time of more than 11 days. Allowing the default to be used effectively means that triggering is disabled after the first trigger message. However, an application can enable triggering again by servicing the queue using a command to alter the queue to reset the trigger attribute.

-u *DeadLetterQueue*
The name of the local queue that is to be used as the dead-letter (undelivered-message) queue. Messages are put on this queue if they cannot be routed to their correct destination.

The default is no dead-letter queue.

-x *MaximumUncommittedMessages*
The maximum number of uncommitted messages under any one sync point. The uncommitted messages are the sum of:

- The number of messages that can be retrieved from queues
- The number of messages that can be put on queues
- Any trigger messages generated within this unit of work

This limit does not apply to messages that are retrieved or put outside a sync point.

Specify a value in the range 1 - 999999999. The default value is 10000 uncommitted messages.

-z Suppresses error messages.

This flag is used within IBM WebSphere MQ to suppress unwanted error messages. Do not use this flag when using a command line. Using this flag can result in a loss of information.

Return codes

Return code	Description
0	Queue manager created
8	Queue manager exists
39	Invalid parameter specified
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage unavailable
70	Queue space unavailable
71	Unexpected error
72	Queue manager name error
74	The IBM WebSphere MQ service is not started.

Return code	Description
100	Log location invalid
111	Queue manager created. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification might be incorrect.
115	Invalid log size
119	Permission denied (Windows only)

Examples

- The following command creates a default queue manager called `Paint.queue.manager`, with a description of `Paint shop`, and creates the system and default objects. It also specifies that linear logging is to be used:

  ```
  crtmqm -c "Paint shop" -ll -q Paint.queue.manager
  ```

- The following command creates a default queue manager called `Paint.queue.manager`, creates the system and default objects, and requests two primary and three secondary log files:

  ```
  crtmqm -c "Paint shop" -ll -lp 2 -ls 3 -q Paint.queue.manager
  ```

- The following command creates a queue manager called `travel`, creates the system and default objects, sets the trigger interval to 5000 milliseconds (5 seconds), and specifies `SYSTEM.DEAD.LETTER.QUEUE` as its dead-letter queue.

  ```
  crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE travel
  ```

- The following command creates a queue manager called `QM1` on UNIX and Linux systems, which has log and queue manager data folders in a common parent directory. The parent directory is to be shared on highly available networked storage to create a multi-instance queue manager. Before issuing the command, create other parameters `/MQHA`, `/MQHA/logs` and `/MQHA/qmgrs` owned by the user and group `mqm`, and with permissions `rwxrwxr-x`.

  ```
  crtmqm -ld /MQHA/logs -md /MQHA/qmgrs QM1
  ```

Related commands

Command	Description
strmqm	Start queue manager
endmqm	End queue manager
dltmqm	Delete queue manager
setmqm	Set associated installation

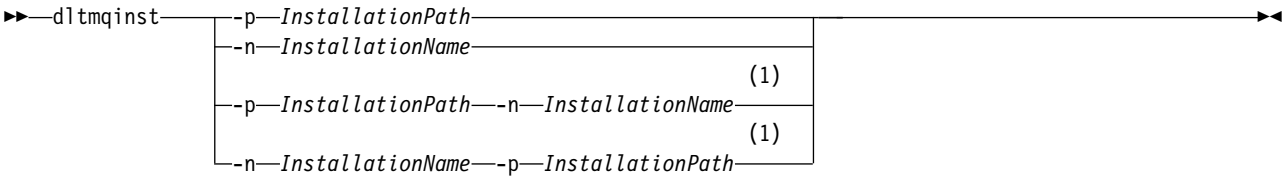
dltmqinst:

Delete installation entries from `mqinst.ini` on UNIX and Linux systems.

Purpose

File `mqinst.ini` contains information about all IBM WebSphere MQ installations on a system. For more information about mqinst.ini, see Installation configuration file, mqinst.ini .

Syntax



Notes:

1 When specified together, the installation name and installation path must refer to the same installation.

Parameters

-n *InstallationName*
The name of the installation.

-p *InstallationPath*
The installation path is the location where IBM WebSphere MQ is installed.

Return codes

Return code	Description
0	Entry deleted without error
5	Entry still active
36	Invalid arguments supplied
44	Entry does not exist
59	Invalid installation specified
71	Unexpected error
89	ini file error
96	Could not lock ini file
98	Insufficient authority to access ini file
131	Resource problem

Example

1. This command deletes an entry with an installation name of `myInstallation`, and an installation path of `/opt/myInstallation`:

```
dltmqinst -n MyInstallation -p /opt/myInstallation
```

Note: You can only use the **dltmqinst** command on another installation from the one it runs from. If you only have one IBM WebSphere MQ installation, the command will not work.

Note: On a Solaris 10 MQ Client installation, only the root user has permissions to edit the mqinst.ini file.

dltmqm:

Delete a queue manager.

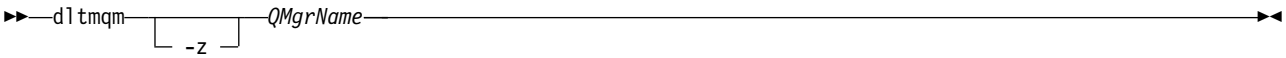
Purpose

Use the **dltmqm** command to delete a specified queue manager and all objects associated with it. Before you can delete a queue manager, you must end it using the **endmqm** command.

You must use the **dltmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

In WebSphere MQ for Windows, it is an error to delete a queue manager when queue manager files are open. If you get this error, close the files and reissue the command.

Syntax



Required parameters

QMgrName
The name of the queue manager to delete.

Optional parameters

-z Suppresses error messages.

Return codes

Return code	Description
0	Queue manager deleted
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
24	A process that was using the previous instance of the queue manager has not yet disconnected.
25	An error occurred while creating or checking the directory structure for the queue manager.
26	Queue manager running as a standby instance.
27	Queue manager could not obtain data lock.
29	Queue manager deleted, however there was a problem removing it from Active Directory.
33	An error occurred while deleting the directory structure for the queue manager.
49	Queue manager stopping
58	Inconsistent use of installations detected
62	The queue manager is associated with a different installation
69	Storage not available
71	Unexpected error
72	Queue manager name error
74	The WebSphere MQ service is not started.
100	Log location invalid.
112	Queue manager deleted. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification might be incorrect.
119	Permission denied (Windows only).

Examples

1. The following command deletes the queue manager `saturn.queue.manager`.

   ```
   dltmqm saturn.queue.manager
   ```

2. The following command deletes the queue manager `travel` and also suppresses any messages caused by the command.

   ```
   dltmqm -z travel
   ```

Usage notes

In WebSphere MQ for Windows, it is an error to delete a queue manager when queue manager files are open. If you get this error, close the files and reissue the command.

Deleting a cluster queue manager does not remove it from the cluster. To check whether the queue manager you want to delete is part of a cluster, issue the command **DIS CLUSQMGR(*)**. Then check whether this queue manager is listed in the output. If it is listed as a cluster queue manager you must remove the queue manager from the cluster before deleting it. See the related link for instructions.

If you do delete a cluster queue manager without first removing it from the cluster, the cluster continues to regard the deleted queue manager as a member of the cluster for at least 30 days. You can remove it from the cluster using the command **RESET CLUSTER** on a full repository queue manager. Re-creating a queue manager with an identical name and then trying to remove that queue manager from the cluster does not result in the cluster queue manager being removed from the cluster. This is because the newly created queue manager, although having the same name, does not have the same queue manager ID (QMID). Therefore it is treated as a different queue manager by the cluster.

Related commands

Command	Description
crtmqm	Create queue manager
strmqm	Start queue manager
endmqm	End queue manager

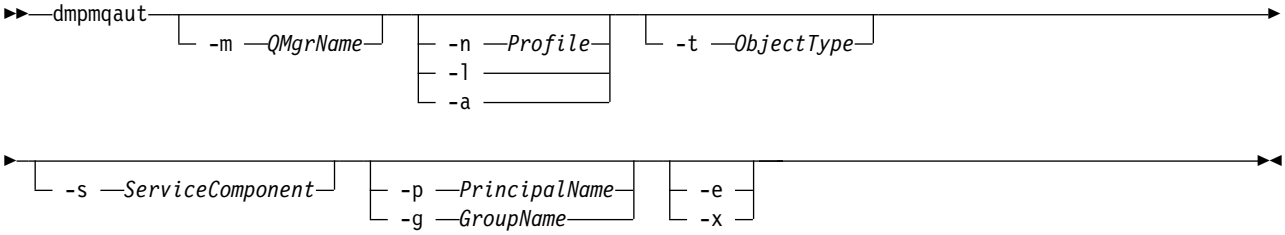
dmpmqaut:

Dump a list of current authorizations for a range of WebSphere MQ object types and profiles.

Purpose

Use the **dmpmqaut** command to dump the current authorizations to a specified object.

Syntax



Optional parameters

-m *QMgrName*
Dump authority records only for the queue manager specified. If you omit this parameter, only authority records for the default queue manager are dumped.

-n *Profile*
The name of the profile for which to dump authorizations. The profile name can be generic, using wildcard characters to specify a range of names as explained in Using OAM generic profiles on UNIX or Linux systems and Windows.

-l
Dump only the profile name and type. Use this option to generate a *terse* list of all defined profile names and types.

-a Generate set authority commands.

-t *ObjectType*
The type of object for which to dump authorizations. Possible values are:

Value	Description
authinfo	An authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or **chl**	A channel
clntconn or **clcn**	A client connection channel
listener or **lstr**	A listener
namelist or **nl**	A namelist
process or **prcs**	A process
queue or **q**	A queue or queues matching the object name parameter
qmgr	A queue manager
rqmname or **rqmn**	A remote queue manager name
service or **srvc**	A service
topic or **top**	A topic

-s *ServiceComponent*
If installable authorization services are supported, specifies the name of the authorization service for which to dump authorizations. This parameter is optional; if you omit it, the authorization inquiry is made to the first installable component for the service.

-p *PrincipalName*
This parameter applies to WebSphere MQ for Windows only; UNIX systems keep only group authority records.

The name of a user for whom to dump authorizations to the specified object. The name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see Principals and groups.

-g *GroupName*
The name of the user group for which to dump authorizations. You can specify only one name, which must be the name of an existing user group.

For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

-e Display all profiles used to calculate the cumulative authority that the entity has to the object specified in `-n Profile`. The variable *Profile* must not contain any wildcard characters.

The following parameters must also be specified:

- `-m QMgrName`
- `-n Profile`
- `-t ObjectType`

and either -p *PrincipalName*, or -g *GroupName*.

-x Display all profiles with the same name as specified in -n *Profile*. This option does not apply to the **QMGR** object, so a dump request of the form dmpmqaut -m QM -t QMGR ... -x is not valid.

Examples

The following examples show the use of dmpmqaut to dump authority records for generic profiles:

1. This example dumps all authority records with a profile that matches queue a.b.c for principal user1.

   ```
   dmpmqaut -m qm1 -n a.b.c -t q -p user1
   ```

   The resulting dump would look something like this:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   ```

   Note: UNIX users cannot use the -p option; they must use -g groupname instead.

2. This example dumps all authority records with a profile that matches queue a.b.c.

   ```
   dmpmqaut -m qmgr1 -n a.b.c -t q
   ```

   The resulting dump would look something like this:

   ```
   profile:     a.b.c
   object type: queue
   entity:      Administrator
   type:        principal
   authority:   all
   - - - - - - - - - - - - - - - - -
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   - - - - - - - - - - - - - - - - -
   profile:     a.**
   object type: queue
   entity:      group1
   type:        group
   authority:   get
   ```

3. This example dumps all authority records for profile a.b.*, of type queue.

   ```
   dmpmqaut -m qmgr1 -n a.b.* -t q
   ```

   The resulting dump would look something like this:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1
   type:        principal
   authority:   get, browse, put, inq
   ```

4. This example dumps all authority records for queue manager qmX.

   ```
   dmpmqaut -m qmX
   ```

   The resulting dump would look something like this:

   ```
   profile:     q1
   object type: queue
   entity:      Administrator
   type:        principal
   authority:   all
   - - - - - - - - - - - - - - - - -
   ```

```
profile:     q*
object type: queue
entity:      user1
type:        principal
authority:   get, browse
- - - - - - - - - - - - - - - - -
profile:     name.*
object type: namelist
entity:      user2
type:        principal
authority:   get
- - - - - - - - - - - - - - - - -
profile:     pr1
object type: process
entity:      group1
type:        group
authority:   get
```

5. This example dumps all profile names and object types for queue manager qmX.

   ```
   dmpmqaut -m qmX -l
   ```

   The resulting dump would look something like this:

   ```
   profile: q1, type: queue
   profile: q*, type: queue
   profile: name.*, type: namelist
   profile: pr1, type: process
   ```

Note:

1. For WebSphere MQ for Windows only, all principals displayed include domain information, for example:

   ```
   profile:     a.b.*
   object type: queue
   entity:      user1@domain1
   type:        principal
   authority:   get, browse, put, inq
   ```

2. Each class of object has authority records for each group or principal. These records have the profile name `@CLASS` and track the `crt` (create) authority common to all objects of that class. If the `crt` authority for any object of that class is changed then this record is updated. For example:

   ```
   profile:     @class
   object type: queue
   entity:      test
   entity type: principal
   authority:   crt
   ```

   This shows that members of the group `test` have `crt` authority to the class `queue`.

3. For WebSphere MQ for Windows only, members of the "Administrators" group are by default given full authority. This authority, however, is given automatically by the OAM, and is not defined by the authority records. The **dmpmqaut** command displays authority defined only by the authority records. Unless an authority record has been explicitly defined, therefore, running the **dmpmqaut** command against the "Administrators" group displays no authority record for that group.

dmpmqcfg:

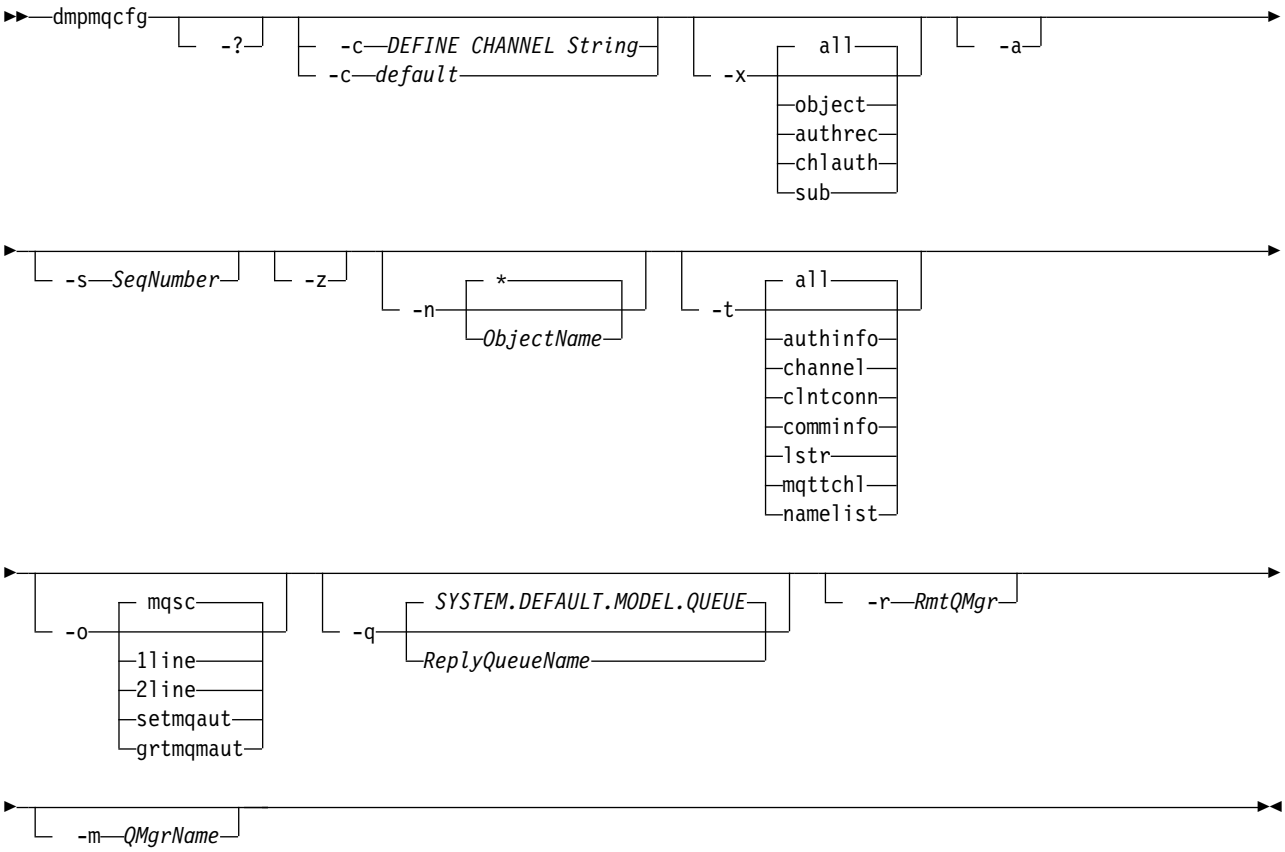
Use the **dmpmqcfg** command to dump the configuration of a WebSphere MQ queue manager.

Purpose

Use the **dmpmqcfg** command to dump the configuration of WebSphere MQ queue managers. If any default object has been edited, the -a option must be used if the dumped configuration will be used to restore the configuration.

The **dmpmqcfg** utility dumps only subscriptions of type MQSUBTYPE_ADMIN, that is, only subscriptions that are created using the MQSC command **DEFINE SUB** or its PCF equivalent. The output from **dmpmqcfg** is a **runmqsc** command to enable the administration subscription to be re-created. Subscriptions that are created by applications using the MQSUB MQI call of type MQSUBTYPE_API are not part of the queue manager configuration, even if durable, and so are not dumped by **dmpmqcfg**. MQTT channels will only be returned for types -t all and -t mqttchl if the telemetry (MQXR) service is running. For instructions on how to start the telemetry service, see Administering IBM WebSphere MQ Telemetry.

Note: The **dmpmqcfg** command does not make a backup of the IBM WebSphere MQ Advanced Message Security policies. If you want to export the IBM WebSphere MQ Advanced Message Security policies, ensure that you run **dspmqspl** with the ¬export flag. This command exports the policies for IBM WebSphere MQ Advanced Message Security into a text file, which can be used for restoration purposes. For more information see "dspmqspl" on page 181.



Optional Parameters

-? Inquire the usage message for dmpmqcfg.

-c Force a client mode connection. If the **-c** parameter is qualified with the option default, the default client connection process is used. If **-c** is omitted, the default is to attempt to connect to the queue manager first by using server bindings and then if this fails by using client bindings.

If the option is qualified with an MQSC DEFINE CHANNEL CHLTYPE(CLNTCONN) string then this is parsed and if successful, used to create a temporary connection to the queue manager.

-x *[all | object | authrec | chlauth | sub]*
Filter the definition procedure to show object definitions, authority records, channel authentication records, or durable subscriptions . The default value all is that all types are returned.

-a Return object definitions to show all attributes. The default is to return only attributes which differ from the defaults for the object type.

-s*SeqNumber*
Reset channel sequence number for sender, server and cluster sender channel types to the numeric value specified. The value `SeqNumber` must be in the range 1 - 999999999.

-z Activate silent mode in which warnings, such as those which appear when inquiring attributes from a queue manager of a higher command level are suppressed.

-n *[* | ObjectName]*
Filter the definitions produced by object or profile name, the object/profile name may contain a single asterisk. The `*` option can be placed only at the end of the entered filter string.

-t
Choose a single type of object for which to export. Possible values are:

Value	Description
`all`	All object types
`authinfo`	An authentication information object
`channel` or `chl`	A channel
`comminfo`	A communications information object
`lstr` or `listener`	A listener
`mqttchl`	An MQTT channel
`namelist` or `nl`	A namelist
`process` or `prcs`	A process
`queue` or `q`	A queue
`qmgr`	A queue manager
`srvc` or `service`	A service
`topic` or `top`	A topic

-o*[mqsc | 1line | 2line | setmqaut | grtmqmaut]*
Possible values are:

Value	Description
`mqsc`	Multi-line MQSC that can be used as direct input to **runmqsc**
`1line`	MQSC with all attributes on a single line for line diffing
`2line`	MQSC with output on two lines. The first line is an MQSC command string and the second is a commented version with immutable values.
`setmqaut`	setmqaut statements for UNIX and Windows queue managers; valid only when `-x authrec` is specified
`grtmqmaut`	Linux only; generates iSeries syntax for granting access to the objects.

Note: If you wish to use the option `2line`, you must ensure that you have applied APAR IT00612 to your IBM WebSphere MQ Version 7.5 installation.

-q The name of the reply-to queue used when getting configuration information.

-r The name of the remote queue manager/transmit queue when using queued mode. If this parameter is omitted the configuration for the directly connected queue manager (specified with the **-m** parameter) is dumped.

-m The name of the queue manager to connect to. If omitted the default queue manager name is used.

Authorizations

The user must have MQZAO_OUTPUT (+put) authority to access the command input queue (SYSTEM.ADMIN.COMMAND.QUEUE) and MQZAO_DISPLAY (+dsp) authority to access the default model queue (SYSTEM.DEFAULT.MODEL.QUEUE), to be able to create a temporary dynamic queue if using the default reply queue.

The user must also have MQZAO_CONNECT (+connect) and MQZAO_INQUIRE (+inq) authority for the queue manager, and MQZAO_DISPLAY (+dsp) authority for every object that is requested.

Return code

If a failure occurs **dmpmqcfg** returns an error code. Otherwise, the command outputs a footer, an example of which follows:

```
********************************************************************************
* Script ended on 2016-01-05   at 05.10.09
* Number of Inquiry commands issued: 14
* Number of Inquiry commands completed: 14
* Number of Inquiry responses processed: 273
* QueueManager count: 1
* Queue count: 55
* NameList count: 3
* Process count: 1
* Channel count: 10
* AuthInfo count: 4
* Listener count: 1
* Service count: 1
* CommInfo count: 1
* Topic count: 5
* Subscription count: 1
* ChlAuthRec count: 3
* Policy count: 1
* AuthRec count: 186
* Number of objects/records: 273
********************************************************************************
```

Examples

To make these examples work you need to ensure that your system is set up for remote MQSC operation. See Preparing queue managers for remote administration and Preparing channels and transmission queues for remote administration.

```
dmpmqcfg -m MYQMGR -c "DEFINE CHANNEL(SYSTEM.ADMIN.SVRCONN) CHLTYPE(CLNTCONN)
         CONNAME('myhost.mycorp.com(1414)')"
```

dumps all the configuration information from remote queue manager *MYQMGR* in MQSC format and creates an ad-hoc client connection to the queue manager using a client channel called *SYSTEM.ADMIN.SVRCONN*.

Note: You need to ensure that a server-connection channel with the same name exists.

```
dmpmqcfg -m LOCALQM -r MYQMGR
```

dumps all configuration information from remote queue manager *MYQMGR*, in MQSC format, connects initially to local queue manager *LOCALQM*, and sends inquiry messages through this local queue manager.

Note: You need to ensure that the local queue manager has a transmission queue named *MYQMGR*, with channel pairings defined in both directions, to send and receive replies between queue managers.

Related concepts:

Restoring queue manager configuration
Follow these steps to restore a backup of a queue manager's configuration.

dmpmqlog:

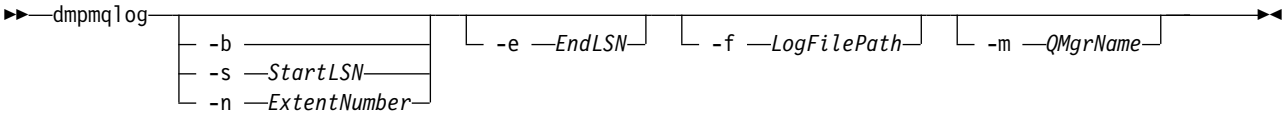
Display and format a portion of the WebSphere MQ system log.

Purpose

Use the **dmpmqlog** command to dump a formatted version of the WebSphere MQ system log to standard out.

The log to be dumped must have been created on the same type of operating system as that being used to issue the command.

Syntax



Optional parameters

Dump start point
Use one of the following parameters to specify the log sequence number (LSN) at which the dump should start. If you omit this, dumping starts by default from the LSN of the first record in the active portion of the log.

-b Start dumping from the base LSN. The base LSN identifies the start of the log extent that contains the start of the active portion of the log.

-s *StartLSN*
Start dumping from the specified LSN. The LSN is specified in the format `nnnn:nnnn:nnnn:nnnn`.

If you are using a circular log, the LSN value must be equal to or greater than the base LSN value of the log.

-n *ExtentNumber*
Start dumping from the specified extent number. The extent number must be in the range 0 - 9999999.

This parameter is valid only for queue managers using linear logging.

-e *EndLSN*
End dumping at the specified LSN. The LSN is specified in the format `nnnn:nnnn:nnnn:nnnn`.

-f *LogFilePath*
The absolute (rather than relative) directory path name to the log files. The specified directory must contain the log header file (`amqhlctl.lfh`) and a subdirectory called `active`. The active subdirectory must contain the log files. By default, log files are assumed to be in the directories specified in the WebSphere MQ configuration information. If you use this option, queue names associated with queue identifiers are shown in the dump only if you use the -m option to name a queue manager name that has the object catalog file in its directory path.

On a system that supports long file names this file is called qmqmobjcat and, to map the queue identifiers to queue names, it must be the file used when the log files were created. For example, for a queue manager named qm1, the object catalog file is located in the directory `..\qmgrs\qm1\`

qmanager\. To achieve this mapping, you might need to create a temporary queue manager, for example named tmpq, replace its object catalog with the one associated with the specific log files, and then start dmpmqlog, specifying `-m tmpq` and `-f` with the absolute directory path name to the log files.

-m *QMgrName*
The name of the queue manager. If you omit this parameter, the name of the default queue manager is used.

Note: Do not dump the log while the queue manager is running, and do not start the queue manager while **dmpmqlog** is running.

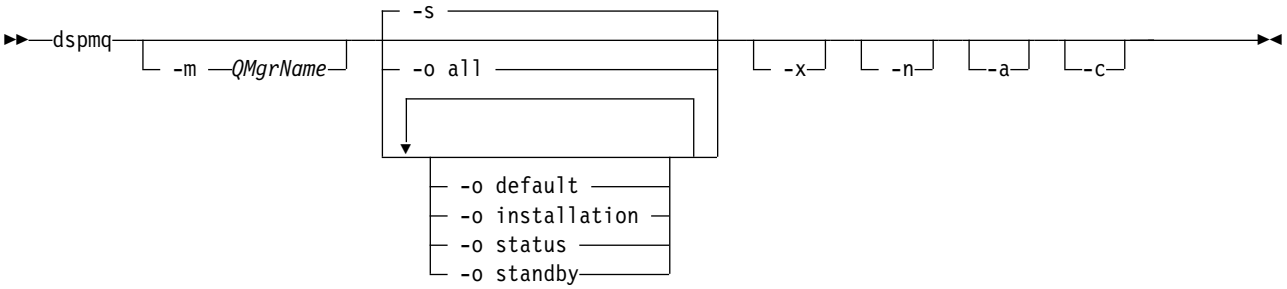
dspmq:

Display information about queue managers.

Purpose

Use the **dspmq** command to display names and details of the queue managers on a system.

Syntax



Required parameters

None

Optional parameters

-a Displays information about the active queue managers only.

A queue manager is active if it is associated with the installation from which the **dspmq** command was issued and one or more of the following statements are true:

- The queue manager is running
- A listener for the queue manager is running
- A process is connected to the queue manager

-m *QMgrName*
The queue manager for which to display details. If you give no name, all queue manager names are displayed.

-n Suppresses translation of output strings.

-s
The operational status of the queue managers is displayed. This parameter is the default status setting.

The parameter *-o status* is equivalent to *-s*.

-o all
The operational status of the queue managers is displayed, and whether any are the default queue manager.

On Windows, UNIX and Linux, the installation name (INSTNAME), installation path (INSTPATH), and installation version (INSTVER) of the installation that the queue manager is associated with is also displayed.

-o default
Displays whether any of the queue managers are the default queue manager.

-o installation
Windows, UNIX and Linux only.

Displays the installation name (INSTNAME), installation path (INSTPATH), and installation version (INSTVER) of the installation that the queue manager is associated with.

-o status
The operational status of the queue managers is displayed.

-o standby
Displays whether a queue manager currently permits starting a standby instance. The possible values are shown in Table 40.

Table 40. Standby values

Value	Description
Permitted	The queue manager is running and is permitting standby instances.
Not permitted	The queue manager is running and is not permitting standby instances.
Not applicable	The queue manager is not running. You can start the queue manager and this instance becomes active if it starts successfully.

-x Information about queue manager instances are displayed. The possible values are shown in Table 41.

Table 41. Instance values

Value	Description
Active	The instance is the active instance.
Standby	The instance is a standby instance.

-c Shows the list of processes currently connected to the IPCC, QMGR, and PERSISTENT sub pools for a queue manager.

For example, this list typically includes:

- Queue manager processes
- Applications, including those that are inhibiting shutdown
- Listeners

Queue Manager States

The following is a list of the different states a queue manager can be in:

Starting
Running
Running as standby
Running elsewhere
Quiescing
Ending immediately
Ending pre-emptively
Ended normally
Ended immediately
Ended unexpectedly
Ended pre-emptively
Status not available

Return codes

Return code	Description
0	Command completed normally
36	Invalid arguments supplied
58	Inconsistent use of installations detected
71	Unexpected error
72	Queue manager name error

Examples

1. The following command displays queue managers on this server:
   ```
   dspmq -o all
   ```
2. The following command displays standby information for queue managers on this server that have ended immediately:
   ```
   dspmq -o standby
   ```
3. The following command displays standby information and instance information for queue managers on this server:
   ```
   dspmq -o standby -x
   ```

dspmqaut:

dspmqaut displays the authorizations of a specific WebSphere MQ object.

Purpose

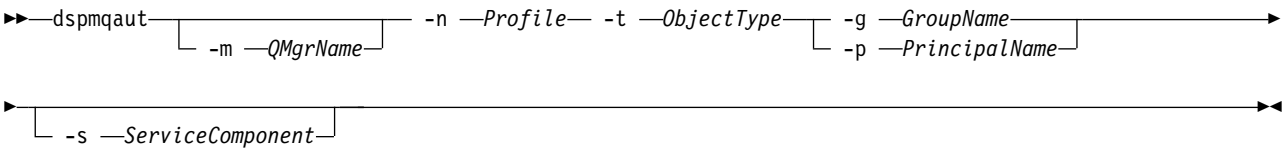
Use the **dspmqaut** command to display the current authorizations to a specified object.

If a user ID is a member of more than one group, this command displays the combined authorizations of all the groups.

Only one group or principal can be specified.

For more information about authorization service components, see Installable services, Service components , and Authorization service interface.

Syntax



Required parameters

-n *Profile*
The name of the profile for which to display authorizations. The authorizations apply to all WebSphere MQ objects with names that match the profile name specified.

This parameter is required, unless you are displaying the authorizations of a queue manager. In this case you must not include it and instead specify the queue manager name using the **-m** parameter.

-t *ObjectType*
The type of object on which to make the inquiry. Possible values are:

Table 42. Object Types

Object Type Command	Object Description
authinfo	An authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or **chl**	A channel
clntconn or **clcn**	A client connection channel
listener or **lstr**	A Listener
namelist or **nl**	A namelist
process or **prcs**	A process
queue or **q**	A queue or queues matching the object name parameter
rqmname or **rqmn**	A remote queue manager name
service or **srvc**	A service
topic or **top**	A topic

Optional parameters

-m *QMgrName*
The name of the queue manager on which to make the inquiry. This parameter is optional if you are displaying the authorizations of your default queue manager.

-g *GroupName*
The name of the user group on which to make the inquiry. You can specify only one name, which must be the name of an existing user group.

For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

-p *PrincipalName*
The name of a user for whom to display authorizations to the specified object.

For WebSphere MQ for Windows only, the name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see Principals and groups.

-s *ServiceComponent*
If installable authorization services are supported, specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if you omit it, the authorization inquiry is made to the first installable component for the service.

Returned parameters

Returns an authorization list, which can contain none, one, or more authorization values. Each authorization value returned means that any user ID in the specified group or principal has the authority to perform the operation defined by that value.

Table 43 shows the authorities that can be given to the different object types.

Table 43. Specifying authorities for different object types

Authority	Queue	Process	Queue manager	Remote queue manager name	Namelist	Topic	Auth info	Clntconn	Channel	Listener	Service
all	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
alladm	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
allmqi	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No
none	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No	No	No	No	No	No	No
browse	Yes	No	No	No	No	No	No	No	No	No	No
chg	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
clr	Yes	No	No	No	No	Yes	No	No	No	No	No
connect	No	No	Yes	No	No	No	No	No	No	No	No
crt	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ctrl	No	No	No	No	No	Yes	No	No	Yes	Yes	Yes
ctrlx	No	No	No	No	No	No	No	No	Yes	No	No
dlt	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
get	Yes	No	No	No	No	No	No	No	No	No	No
pub	No	No	No	No	No	Yes	No	No	No	No	No
put	Yes	No	No	Yes	No	Yes	No	No	No	No	No
inq	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No
passall	Yes	No	No	No	No	Yes	No	No	No	No	No
passid	Yes	No	No	No	No	Yes	No	No	No	No	No
resume	No	No	No	No	No	Yes	No	No	No	No	No
set	Yes	Yes	Yes	No	No	No	No	No	No	No	No
setall	Yes	No	Yes	No	No	Yes	No	No	No	No	No
setid	Yes	No	Yes	No	No	Yes	No	No	No	No	No
sub	No	No	No	No	No	Yes	No	No	No	No	No
system	No	No	Yes	No	No	No	No	No	No	No	No

The following list defines the authorizations associated with each value:

Table 44. Authorizations associated with values.

Authorization Commands	Description
all	Use all operations relevant to the object. `all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.
alladm	Perform all administration operations relevant to the object
allmqi	Use all MQI calls relevant to the object
altusr	Specify an alternative user ID on an MQI call
browse	Retrieve a message from a queue by issuing an **MQGET** call with the BROWSE option
chg	Change the attributes of the specified object, using the appropriate command set
clr	Clear a queue (PCF command Clear queue only) or a topic
ctrl	Start, and stop the specified channel, listener, or service, and ping the specified channel.
ctrlx	Reset or resolve the specified channel
connect	Connect the application to the specified queue manager by issuing an **MQCONN** call
crt	Create objects of the specified type using the appropriate command set
dlt	Delete the specified object using the appropriate command set
dsp	Display the attributes of the specified object using the appropriate command set
get	Retrieve a message from a queue by issuing an **MQGET** call
inq	Make an inquiry on a specific queue by issuing an **MQINQ** call
passall	Pass all context
passid	Pass the identity context
pub	Publish a message on a topic using the **MQPUT** call.
put	Put a message on a specific queue by issuing an **MQPUT** call
resume	Resume a subscription using the **MQSUB** call.
set	Set attributes on a queue from the MQI by issuing an **MQSET** call
setall	Set all context
setid	Set the identity context
sub	Create, alter, or resume a subscription to a topic using the **MQSUB** call.
system	Use queue manager for internal system operations

The authorizations for administration operations, where supported, apply to these command sets:

- Control commands

- MQSC commands
- PCF commands

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing

Examples

- The following example shows a command to display the authorizations on queue manager `saturn.queue.manager` associated with user group `staff`:

  ```
  dspmqaut -m saturn.queue.manager -t qmgr -g staff
  ```

  The results from this command are:

  ```
  Entity staff has the following authorizations for object:
          get
          browse
          put
          inq
          set
          connect
          altusr
          passid
          passall
          setid
  ```

- The following example displays the authorities user1 has for queue a.b.c:

  ```
  dspmqaut -m qmgr1 -n a.b.c -t q -p user1
  ```

  The results from this command are:

  ```
  Entity user1 has the following authorizations for object:
          get
          put
  ```

dspmqcsv:

The status of a command server is displayed

Purpose

Use the **dspmqcsv** command to display the status of the command server for the specified queue manager.

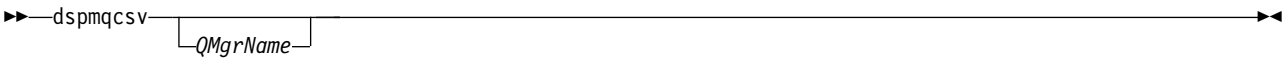
The status can be one of the following:

- Starting

- Running
- Running with SYSTEM.ADMIN.COMMAND.QUEUE not enabled for gets
- Ending
- Stopped

You must use the **dspmqcsv** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

Syntax



Required parameters

None

Optional parameters

QMgrName
The name of the local queue manager for which the command server status is being requested.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command displays the status of the command server associated with `venus.q.mgr`:

```
dspmqcsv venus.q.mgr
```

Related commands

Command	Description
strmqcsv	Start a command server
endmqcsv	End a command server

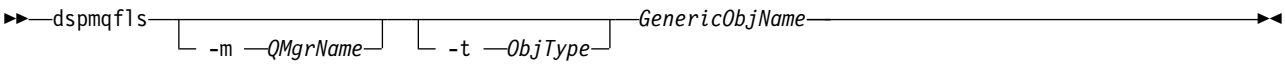
dspmqfls:

Display the file names corresponding to WebSphere MQ objects.

Purpose

Use the **dspmqfls** command to display the real file system name for all WebSphere MQ objects that match a specified criterion. You can use this command to identify the files associated with a particular object. This command is useful for backing up specific objects. See Understanding WebSphere MQ file names for information about name transformation.

Syntax



Required parameters

GenericObjName
The name of the object. The name is a string with no flag and is a required parameter. Omitting the name returns an error.

This parameter supports an asterisk (*) as a wildcard at the end of the string.

Optional parameters

-m *QMgrName*
The name of the queue manager for which to examine files. If you omit this name, the command operates on the default queue manager.

-t *ObjType*
The object type. The following list shows the valid object types. The abbreviated name is shown first followed by the full name.

Table 45. Valid object types

Object Type	Description
* or **all**	All object types; this parameter is the default
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or **chl**	A channel
clntconn or **clcn**	A client connection channel
catalog or **ctlg**	An object catalog
namelist or **nl**	A namelist
listener or **lstr**	A listener
process or **prcs**	A process
queue or **q**	A queue or queues matching the object name parameter
qalias or **qa**	An alias queue
qlocal or **ql**	A local queue
qmodel or **qm**	A model queue
qremote or **qr**	A remote queue
qmgr	A queue manager object
service or **srvc**	A service

Note:

1. The **dspmqfls** command displays the name of the directory containing the queue, ***not*** the name of the queue itself.
2. In WebSphere MQ for UNIX systems, you must prevent the shell from interpreting the meaning of special characters, for example, an asterisk (*). The way you do this depends on the shell you are using. It may involve the use of single quotation marks, double quotation marks, or a backslash.

Return codes

Return code	Description
0	Command completed normally
10	Command completed but not entirely as expected
20	An error occurred during processing

Examples

1. The following command displays the details of all objects with names beginning `SYSTEM.ADMIN` defined on the default queue manager.

   ```
   dspmqfls SYSTEM.ADMIN*
   ```

2. The following command displays file details for all processes with names beginning `PROC` defined on queue manager `RADIUS`.

   ```
   dspmqfls -m RADIUS -t prcs PROC*
   ```

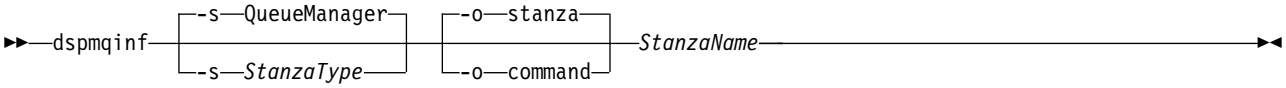
dspmqinf:

Display WebSphere MQ configuration information (Windows and UNIX platforms only).

Purpose

Use the **dspmqinf** command to display WebSphere MQ configuration information.

Syntax



Required parameters

StanzaName
The name of the stanza. That is, the value of the key attribute that distinguishes between multiple stanzas of the same type.

Optional parameters

-s *StanzaType*
The type of stanza to display. If omitted, the QueueManager stanza is displayed.

The only supported value of *StanzaType* is `QueueManager`.

-o stanza
Displays the configuration information in stanza format as it is shown in the `.ini` files. This format is the default output format.

Use this format to display stanza information in a format that is easy to read.

-o command
Displays the configuration information as an **addmqinf** command.

Information about the installation associated with the queue manager is not displayed using this parameter. The **addmqinf** command does not require information about the installation.

Use this format to paste into a command shell.

Return codes

Return code	Description
0	Successful operation
39	Bad command-line parameters
44	Stanza does not exist
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error

Examples

```
dspmqinf QM.NAME
```

The command defaults to searching for a QueueManager stanza named `QM.NAME` and displays it in stanza format.

```
QueueManager:
   Name=QM.NAME
   Prefix=/var/mqm
   Directory=QM!NAME
   DataPath=/MQHA/qmgrs/QM!NAME
   InstallationName=Installation1
```

The following command gives the same result:

```
dspmqinf -s QueueManager -o stanza QM.NAME
```

The next example displays the output in **addmqinf** format.

```
dspmqinf -o command QM.NAME
```

The output is on one line:

```
addmqinf -s QueueManager -v Name=QM.NAME -v Prefix=/var/mqm -v Directory=QM!NAME
         -v DataPath=/MQHA/qmgrs/QM!NAME
```

Usage notes

Use **dspmqinf** with **addmqinf** to create an instance of a multi-instance queue manager on a different server.

To use this command you must be a WebSphere MQ administrator and a member of the `mqm` group.

Related commands

Command	Description
"addmqinf" on page 133	Add queue manager configuration information
"rmvmqinf" on page 204	Remove queue manager configuration information

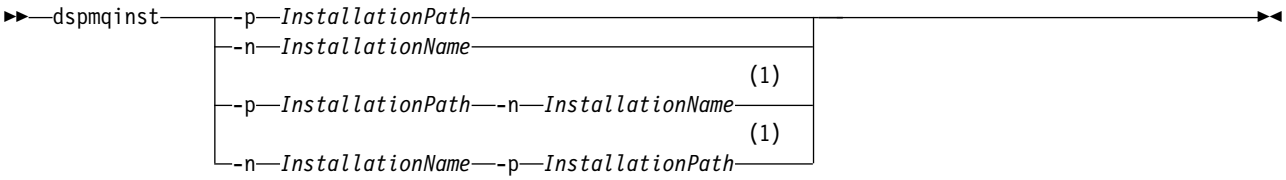
dspmqinst:

Display installation entries from `mqinst.ini` on UNIX, Linux, and Windows.

Purpose

File `mqinst.ini` contains information about all IBM WebSphere MQ installations on a system. For more information about mqinst.ini, see Installation configuration file, mqinst.ini .

Syntax



Notes:

1 When specified together, the installation name and installation path must refer to the same installation.

Parameters

-n *InstallationName*
The name of the installation.

-p *InstallationPath*
The installation path.

? Display usage information.

Return codes

Return code	Description
0	Entry displayed without error
36	Invalid arguments supplied
44	Entry does not exist
59	Invalid installation specified
71	Unexpected error
89	.ini file error
96	Could not lock .ini file
131	Resource problem

Examples

1. Display details of all WebSphere MQ installations on the system:
   ```
   dspmqinst
   ```
2. Query the entry for the installation named *Installation3*:
   ```
   dspmqinst -n Installation3
   ```
3. Query the entry with an installation path of `/opt/mqm`:
   ```
   dspmqinst  -p /opt/mqm
   ```
4. Query the entry for the installation named *Installation3*. Its expected installation path is `/opt/mqm`:
   ```
   dspmqinst -n Installation3 -p /opt/mqm
   ```

dspmqrte:

Determine the route that a message has taken through a queue manager network.

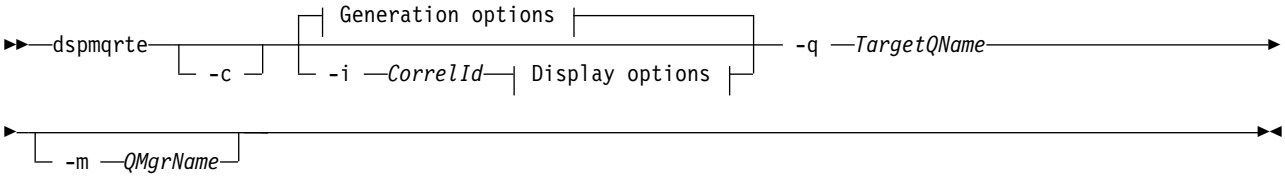
Purpose

The WebSphere MQ display route application (dspmqrte) can be executed on all platforms except z/OS. You can execute the WebSphere MQ display route application as a client to a WebSphere MQ for z/OS queue manager by specifying the -c parameter when issuing the dspmqrte command.

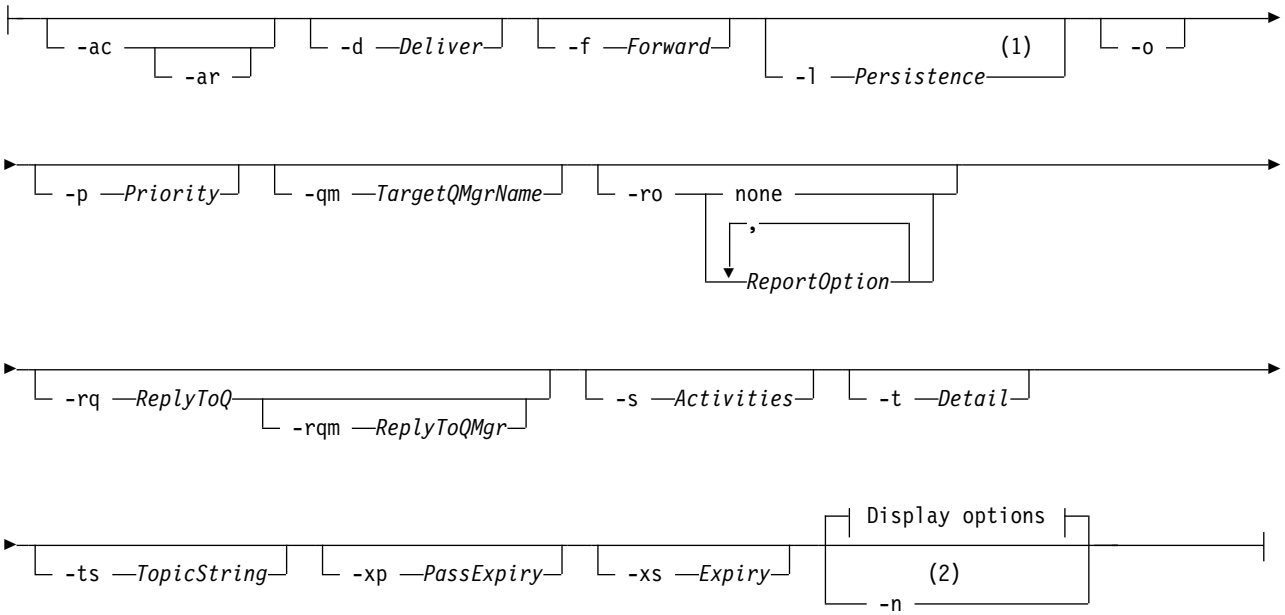
Note: To run a client application against a queue manager, the Client Attachment feature must be installed.

The WebSphere MQ display route application generates and puts a trace-route message into a queue manager network. As the trace-route message travels through the queue manager network, activity information is recorded. When the trace-route message reaches its target queue, the activity information is collected by the WebSphere MQ display route application and displayed. For more information, and examples of using the WebSphere MQ display route application, see WebSphere MQ display route application.

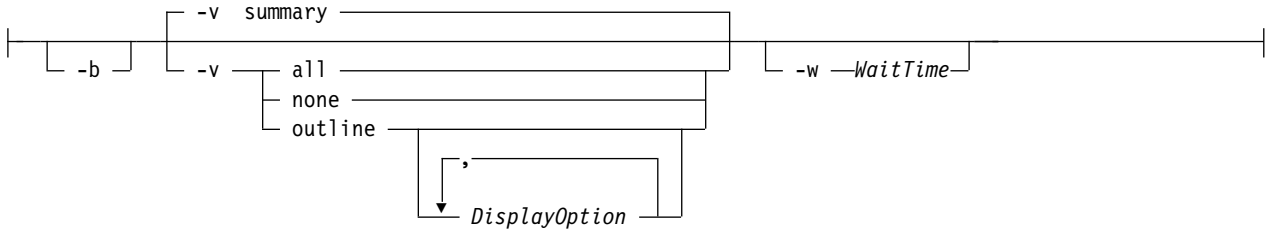
Syntax



Generation options:



Display options:



Notes:

1 If *Persistence* is specified as yes, and is accompanied by a request for a trace-route reply message (*-ar*), or any report generating options (*-ro ReportOption*), then you must specify the parameter *-rq ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.

2 If this parameter is accompanied by a request for a trace-route reply message (**-ar**), or any of the report generating options (**-ro** *ReportOption*), then a specific (non-model) reply-to queue must be specified using **-rq** *ReplyToQ*. By default, activity report messages are requested.

Required parameters

-q *TargetQName*
If the WebSphere MQ display route application is being used to send a trace-route message into a queue manager network, *TargetQName* specifies the name of the target queue.

If the WebSphere MQ display route application is being used to view previously gathered activity information, *TargetQName* specifies the name of the queue where the activity information is stored.

Optional parameters

-c
Specifies that the WebSphere MQ display route application connects as a client application. For more information about how to set up client machines, see Installing a IBM WebSphere MQ client .

This parameter can be used only if the client component is installed.

-i *CorrelId*
This parameter is used when the WebSphere MQ display route application is used to display previously accumulated activity information only. There can be many activity reports and trace-route reply messages on the queue specified by *-q TargetQName*. *CorrelId* is used to identify the activity reports, or a trace-route reply message, related to a trace-route message. Specify the message identifier of the original trace-route message in *CorrelId*.

The format of *CorrelId* is a 48 character hexadecimal string.

-m *QMgrName*
The name of the queue manager to which the WebSphere MQ display route application connects. The name can contain up to 48 characters.

If you do not specify this parameter, the default queue manager is used.

Generation options

The following parameters are used when the WebSphere MQ display route application is used to put a trace-route message into a queue manager network.

-ac
Specifies that activity information is to be accumulated within the trace-route message.

If you do not specify this parameter, activity information is not accumulated within the trace-route message.

-ar
Requests that a trace-route reply message containing all accumulated activity information is generated in the following circumstances:

- The trace-route message is discarded by a WebSphere MQ Version 7.0 queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by a WebSphere MQ Version 7.0 queue manager.
- The number of activities performed on the trace-route message exceeds the value of specified in *-s Activities*.

For more information about trace-route reply messages, see Trace-route reply message reference .

If you do not specify this parameter, a trace-route reply message is not requested.

-d *Deliver*
Specifies whether the trace-route message is to be delivered to the target queue on arrival. Possible values for *Deliver* are:

yes On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging.

no On arrival, the trace-route message is **not** put to the target queue.

If you do not specify this parameter, the trace-route message is **not** put to the target queue.

-f *Forward*

Specifies the type of queue manager that the trace-route message can be forwarded to. Queue managers use an algorithm when determining whether to forward a message to a remote queue manager. For details of this algorithm, see The cluster workload management algorithm. The possible values for *Forward* are:

all The trace-route message is forwarded to any queue manager.
Warning: If forwarded to a WebSphere MQ queue manager before Version 6.0, the trace-route message is not recognized and can be delivered to a local queue despite the value of the *-d Deliver* parameter.

supported The trace-route message is only forwarded to a queue manager that honors the *Deliver* parameter from the *TraceRoute* PCF group.

If you do not specify this parameter, the trace-route message is only forwarded to a queue manager that honors the *Deliver* parameter.

-l *Persistence*

Specifies the persistence of the generated trace-route message. Possible values for *Persistence* are:

yes The generated trace-route message is persistent. (MQPER_PERSISTENT).

no The generated trace-route message is not persistent. (MQPER_NOT_PERSISTENT).

q The generated trace-route message inherits its persistence value from the queue specified by *-q TargetQName*. (MQPER_PERSISTENCE_AS_Q_DEF).

A trace-route reply message, or any report messages, returned shares the same persistence value as the original trace-route message.

If *Persistence* is specified as **yes**, you must specify the parameter *-rq ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.

If you do not specify this parameter, the generated trace-route message is not persistent.

-o Specifies that the target queue is not bound to a specific destination. Typically this parameter is used when the trace-route message is to be put across a cluster. The target queue is opened with option MQOO_BIND_NOT_FIXED.

If you do not specify this parameter, the target queue is bound to a specific destination.

-p *Priority*

Specifies the priority of the trace-route message. The value of *Priority* is either greater than or equal to 0, or MQPRI_PRIORITY_AS_Q_DEF. MQPRI_PRIORITY_AS_Q_DEF specifies that the priority value is taken from the queue specified by *-q TargetQName*.

If you do not specify this parameter, the priority value is taken from the queue specified by *-q TargetQName*.

-qm *TargetQMgrName*

Qualifies the target queue name; normal queue manager name resolution applies. The target queue is specified with *-q TargetQName*.

If you do not specify this parameter, the queue manager to which the WebSphere MQ display route application is connected is used as the reply-to queue manager.

-ro none | *ReportOption*

none Specifies no report options are set.

ReportOption Specifies report options for the trace-route message. Multiple report options can be specified using a comma as a separator. Possible values for *ReportOption* are:

activity The report option MQRO_ACTIVITY is set.

coa The report option MQRO_COA_WITH_FULL_DATA is set.

cod The report option MQRO_COD_WITH_FULL_DATA is set.

exception
The report option MQRO_EXCEPTION_WITH_FULL_DATA is set.

expiration
The report option MQRO_EXPIRATION_WITH_FULL_DATA is set.

discard The report option MQRO_DISCARD_MSG is set.

If *-ro ReportOption* or *-ro none* are not specified, then the MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified.

-rq *ReplyToQ*
Specifies the name of the reply-to queue that all responses to the trace-route message are sent to. If the trace-route message is persistent, or if the *-n* parameter is specified, a reply-to queue must be specified that is not a temporary dynamic queue.

If you do not specify this parameter, the system default model queue, SYSTEM.DEFAULT.MODEL.QUEUE is used as the reply-to queue. Using this model queue causes a temporary dynamic queue, for the WebSphere MQ display route application, to be created.

-rqm *ReplyToQMgr*
Specifies the name of the queue manager where the reply-to queue is located. The name can contain up to 48 characters.

If you do not specify this parameter, the queue manager to which the WebSphere MQ display route application is connected is used as the reply-to queue manager.

-s *Activities*
Specifies the maximum number of recorded activities that can be performed on behalf of the trace-route message before it is discarded. This parameter prevents the trace-route message from being forwarded indefinitely if caught in an infinite loop. The value of *Activities* is either greater than or equal to 1, or MQROUTE_UNLIMITED_ACTIVITIES. MQROUTE_UNLIMITED_ACTIVITIES specifies that an unlimited number of activities can be performed on behalf of the trace-route message.

If you do not specify this parameter, an unlimited number of activities can be performed on behalf of the trace-route message.

-t *Detail*
Specifies the activities that are recorded. The possible values for *Detail* are:

low	Activities performed by user-defined application are recorded only.
medium	Activities specified in **low** are recorded. Additionally, activities performed by MCAs are recorded.
high	Activities specified in **low**, and **medium** are recorded. MCAs do not expose any further activity information at this level of detail. This option is available to user-defined applications that are to expose further activity information only. For example, if a user-defined application determines the route a message takes by considering certain message characteristics, the routing logic can be included with this level of detail.

If you do not specify this parameter, medium level activities are recorded.

-ts *TopicString*
Specifies a topic string to which the WebSphere MQ display route application is to publish a trace-route message, and puts this application into topic mode. In this mode, the application traces all of the messages that result from the publish request.

-xp *PassExpiry*
Specifies whether the report option MQRO_DISCARD_MSG and the remaining expiry time from the trace-route message is passed on to the trace-route reply message. Possible values for *PassExpiry* are:

yes	The report option MQRO_PASS_DISCARD_AND_EXPIRY is specified in the message descriptor of the trace-route message. If a trace-route reply message, or activity reports, are generated for the trace-route message, the MQRO_DISCARD_MSG report option (if specified), and the remaining expiry time are passed on. This parameter is the default value.
no	The report option MQRO_PASS_DISCARD_AND_EXPIRY is **not** specified. If a trace-route reply message is generated for the trace-route message, the discard option and remaining expiry time from the trace-route message are **not** passed on.

If you do not specify this parameter, the MQRO_PASS_DISCARD_AND_EXPIRY report option is not specified in the trace-route message.

-xs *Expiry*
Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-n Specifies that activity information returned for the trace-route message is not to be displayed.

If this parameter is accompanied by a request for a trace-route reply message (**-*ar***), or any of the report generating options from (**-*ro ReportOption***), then a specific (non-model) reply-to queue must be specified using **-*rq ReplyToQ***. By default, activity report messages are requested.

After the trace-route message is put to the specified target queue, a 48 character hexadecimal string is returned containing the message identifier of the trace-route message. The message identifier can be used by the WebSphere MQ display route application to display the activity information for the trace-route message at a later time. This can be done using the *-i CorrelId* parameter.

If you do not specify this parameter, activity information returned for the trace-route message is displayed in the form specified by the *-v* parameter.

Display options

The following parameters are used when the WebSphere MQ display route application is used to display collected activity information.

-b Specifies that the WebSphere MQ display route application only browses activity reports or a trace-route reply message related to a message. This parameter allows activity information to be displayed again at a later time.

If you do not specify this parameter, the WebSphere MQ display route application gets activity reports and deletes them, or a trace-route reply message related to a message.

-v summary | all | none | outline *DisplayOption*

summary The queues that the trace-route message was routed through are displayed.

all All available information is displayed.

none No information is displayed.

outline ***DisplayOption*** Specifies display options for the trace-route message. Multiple display options can be specified using a comma as a separator.

If no values are supplied the subsequent information is displayed:

- The application name
- The type of each operation
- Any operation-specific parameters

Possible values for *DisplayOption* are:

activity All non-PCF group parameters in *Activity* PCF groups are displayed.

identifiers
Values with parameter identifiers MQBACF_MSG_ID or MQBACF_CORREL_ID are displayed. This overrides *msgdelta*.

message
All non-PCF group parameters in *Message* PCF groups are displayed. When this value is specified, you cannot specify *msgdelta*.

msgdelta
All non-PCF group parameters in *Message* PCF groups, that have changed since the last operation, are displayed. When this value is specified, you cannot specify *message*.

operation
All non-PCF group parameters in *Operation* PCF groups are displayed.

traceroute
All non-PCF group parameters in *TraceRoute* PCF groups are displayed.

If you do not specify this parameter, a summary of the message route is displayed.

-w *WaitTime*
Specifies the time, in seconds, that the WebSphere MQ display route application waits for activity reports, or a trace-route reply message, to return to the specified reply-to queue.

If you do not specify this parameter, the wait time is specified as the expiry time of the trace-route message, plus 60 seconds.

Return codes

Return code	Description
0	Command completed normally
10	Invalid arguments supplied
20	An error occurred during processing

Examples

1. The following command puts a trace-route message into a queue manager network with the target queue specified as `TARGET.Q`. Providing queue managers on route are enabled for activity recording, activity reports are generated. Depending on the queue manager attribute, ACTIVREC, activity reports are either delivered to the reply-to queue `ACT.REPORT.REPLY.Q`, or are delivered to a system queue. The trace-route message is discarded on arrival at the target queue.

   ```
   dspmqrte -q TARGET.Q -rq ACT.REPORT.REPLY.Q
   ```

   Providing one or more activity reports are delivered to the reply-to queue, `ACT.REPORT.REPLY.Q`, the WebSphere MQ display route application orders and displays the activity information.
2. The following command puts a trace-route message into a queue manager network with the target queue specified as `TARGET.Q`. Activity information is accumulated within the trace-route message, but activity reports are not generated. On arrival at the target queue, the trace-route message is discarded. Depending on the value of the target queue manager attribute, ROUTEREC, a trace-route reply message can be generated and delivered to either the reply-to queue, `TRR.REPLY.TO.Q`, or to a system queue.

   ```
   dspmqrte -ac -ar -ro discard -rq TRR.REPLY.TO.Q -q TARGET.Q
   ```

   Providing a trace-route reply message is generated, and delivered to the reply-to queue `TRR.REPLY.TO.Q`, the WebSphere MQ display route application orders and displays the activity information that was accumulated in the trace-route message.

For more examples of using the WebSphere MQ display route application and its output, see WebSphere MQ display route application examples.

dspmqspl:

Use the **dspmqspl** command to display a list of all policies and details of a named policy.

Syntax

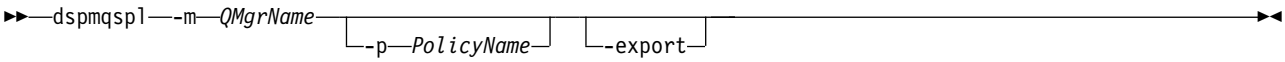


Table 46. `dspmqspl` command flags.

Command flag	Explanation
`-m`	Queue manager name (**mandatory**).
`-p`	Policy name.
`-export`	Adding this flag generates output which can easily applied to a different queue manager.

dspmqtrc:

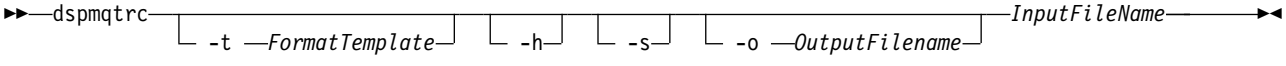
Format and display IBM WebSphere MQ trace.

Purpose

The **dspmqtrc** command is supported on UNIX and HP Integrity NonStop Server systems only. Use the **dspmqtrc** command to display WebSphere MQ formatted trace output.

The runtime SSL trace files have the names AMQ.SSL.TRC and AMQ.SSL.TRC.1. You cannot format any of the SSL trace files. The SSL trace files are binary files and, if they are transferred to IBM support by FTP, they must be transferred in binary transfer mode.

Syntax



Required parameters

InputFileName
The name of the file containing the unformatted trace, for example:
`/var/mqm/trace/AMQ12345.01.TRC`

If you provide one input file, **dspmqtrc** formats it to the output file you name. If you provide more than one input file, any output file you name is ignored, and formatted files are named `AMQ`*`yyyyy.zz`*`.FMT`, based on the PID of the trace file.

Optional parameters

-t *FormatTemplate*
The name of the template file containing details of how to display the trace. If this parameter is not supplied, the default template file location is used:

For AIX systems, the default value is as follows:
MQ_INSTALLATION_PATH`/lib/amqtrc2.fmt`

For all HP Integrity NonStop Server, and UNIX systems other than AIX systems, the default value is as follows:
MQ_INSTALLATION_PATH`/lib/amqtrc.fmt`

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

-h Omit header information from the report.

-s Extract trace header and put to stdout.

-o *output_filename*
The name of the file into which to write formatted data.

Related commands

Command	Description
endmqtrc	End trace
"strmqtrc" on page 247	Start trace

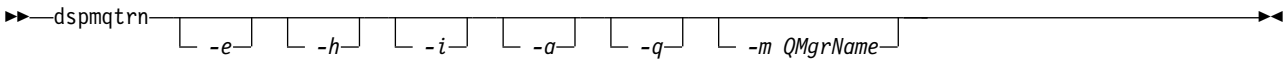
dspmqtrn:

Display in-doubt and heuristically completed transactions.

Purpose

Use the **dspmqtrn** command to display details of transactions. This command includes transactions coordinated by IBM WebSphere MQ and by an external transaction manager.

Syntax



Optional parameters

-e Requests details of externally coordinated, in-doubt transactions. Such transactions are those for which IBM WebSphere MQ has been asked to prepare to commit, but has not yet been informed of the transaction outcome.

-h Requests details of externally coordinated transactions that were resolved by the `rsvmqtrn` command, and the external transaction coordinator has yet to acknowledge with an `xa-forget` command. This transaction state is termed *heuristically completed* by X/Open.

Note: If you do not specify -e, -h, or -i, details of both internally and externally coordinated in-doubt transactions are displayed, but details of externally coordinated, heuristically completed transactions are not displayed.

-i Requests details of internally coordinated, in-doubt transactions. Such transactions are those for which each resource manager has been asked to prepare to commit, but IBM WebSphere MQ has yet to inform the resource managers of the transaction outcome.

Information about the state of the transaction in each of its participating resource managers is displayed. This information can help you assess the affects of failure in a particular resource manager.

Note: If you do not specify -e or -i, details of both internally and externally coordinated in-doubt transactions are displayed.

-a Requests a list of all transactions known to the queue manager . The returned data includes transaction details for all transactions known to the queue manager. If a transaction is currently associated with a IBM WebSphere MQ application connection, information related to that IBM WebSphere MQ application connection is also returned. The data returned by this command might typically be correlated with the output of a runmqsc "DISPLAY CONN" on page 639 command, and the output fields have the same meaning as in that command.

Not all of the fields are appropriate for all transactions. When the fields are not meaningful, they are displayed as blank. For example: The `UOWLOG` value when the command is issued against a circular logging queue manager.

-q Specifying this parameter on its own is the same as specifying `-a -q`.

Displays all the data from the `-a` parameter and a list of up to 100 unique objects updated within the transaction. If more than 100 objects are updated in the same transaction, only the first 100 distinct objects are listed for each transaction.

-m*QMgrName*
The name of the queue manager for which to display transactions. If you omit the name, the transaction of the default queue manager are displayed.

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error
102	No transactions found

Related commands

Command	Description
rsvmqtrn	Resolve transaction

dspmqver:

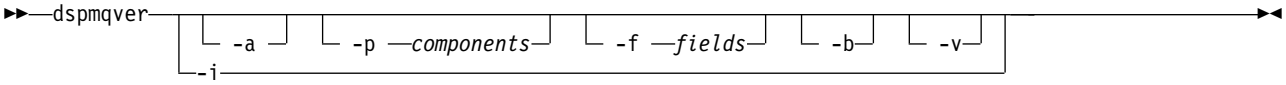
Display WebSphere MQ version and build information.

Purpose

Use the **dspmqver** command to display WebSphere MQ version and build information.

By default, the **dspmqver** command displays details of the installation from which it was invoked. A note is displayed if other installations exist; use the -i parameter to display their details.

Syntax



Optional parameters

-a Display information about all fields and components.

-p *Components*
Display information for the components specified by *component*. Either a single component or multiple components can be specified. Enter either the value of a single component or the sum of the values of all the required components. Available components and related values follow:

1	WebSphere MQ server, or client.
2	WebSphere MQ classes for Java.
4	WebSphere MQ classes for Java Message Service.
8	WebScale Distribution Hub
16[1]	IBM WebSphere MQ custom channel for Windows Communication Foundation
32	IBM Message Service Client for .NET (XMS .NET) - this component is only available on Windows
64	GSKit, or for HP Integrity NonStop Server, SSL

128	WebSphere MQ Advanced Message Security

Notes:

1. Supported by WebSphere MQ for Windows only. If you have not installed Microsoft .NET 3 or later, the following error message is displayed:

   ```
   Title: WMQWCFCustomChannelLevel.exe - Application ErrorThe application failed to initialize properly (0x0000135).
   ```

The default value is 1.

-f *Fields*

Display information for the fields specified by *field*. Specify either a single field or multiple fields. Enter either the value of a single field or the sum of the values of all the required fields. Available fields and related values follow:

1	Name
2	Version, in the form `V.R.M.F`: Where `V`=Version, `R`=Release, `M`=Modification, and `F`=Fix pack
4	Level
8	Build type
16	Platform
32	Addressing mode
64	Operating system
128	Installation path
256[1]	Installation description
512[1]	Installation name
1024[1]	Maximum command level
2048[1]	Primary installation
4096	Data Path

Note:

1. Not applicable to HP Integrity NonStop Server.

Information for each selected field is displayed on a separate line when the dspmqver command is run.

The default value is 8191. This displays information for all fields.

-b Omit header information from the report.

-v Display verbose output.

-i Display information about all installations. You cannot use this option with other options. The installation from which the dspmqver command was issued is displayed first. For any other installations, only the following fields are displayed: Name, Version, Installation name, Installation description, Installation path, and Primary installation. Not applicable to HP Integrity NonStop Server.

Return codes

Return code	Description
0	Command completed normally.
10	Command completed with unexpected results.
20	An error occurred during processing.

Examples

The following command displays WebSphere MQ version and build information, using the default settings for **-p** and **-f** :

```
dspmqver
```

The following command displays information about all fields and components and is the equivalent of specifying `dspmqver -p 63 -f 4095`:

```
dspmqver -a
```

The following command displays version and build information for the WebSphere MQ classes for Java:

```
dspmqver -p 2
```

The following command displays the Common Services for Java Platform Standard Edition, IBM WebSphere MQ, Java Message Service Client, and WebSphere MQ classes for Java Message Service:

```
dspmqver -p 4
```

The following command displays the build level of the WebScale Distribution Hub:

```
dspmqver -p 8 -f 4
```

The following command displays the name and build type for IBM WebSphere MQ custom channel for Windows Communication Foundation:

```
dspmqver -p 16 -f 9
```

The following command displays information about installations of IBM WebSphere MQ.

```
dspmqver -i
```

Command Failure

The **dspmqver** command can fail if you try to view version or build information for the WebSphere MQ classes for Java, and you have not correctly configured your environment. For example, you might see the following message:

```
[root@blade883 ~]# dspmqver -p2
AMQ8351: WebSphere MQ Java environment has not been configured correctly.
```

To resolve this problem, ensure that the path is configured to include the JRE, and that the correct environment variables are set; for example, by using `setjmsenv` or `setjmsenv64`. For example:

```
export PATH=$PATH:/opt/mqm/java/jre/bin
cd /opt/mqm/java/bin/
. ./setjmsenv64

[root@blade883 bin]# dspmqver -p2
Name:       WebSphere MQ classes for Java
Version:    7.1.0.0
Level:      k000-L110908
Build Type: Production
```

endmqcsv:

Stop the command server for a queue manager.

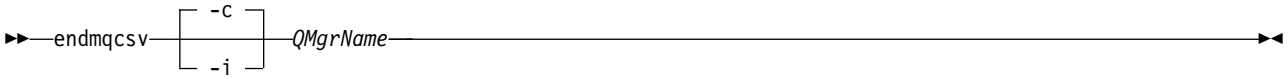
Purpose

Use the **endmqscv** command to stop the command server on the specified queue manager.

You must use the **endmqscv** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

If the queue manager attribute, SCMDSERV, is specified as QMGR then changing the state of the command server using **endmqscv** does not effect how the queue manager acts upon the SCMDSERV attribute at the next restart.

Syntax



Required parameters

QMgrName
The name of the queue manager for which to end the command server.

Optional parameters

-c Stops the command server in a controlled manner. The command server can complete the processing of any command message that it has already started. No new message is read from the command queue.

This parameter is the default.

-i Stops the command server immediately. Actions associated with a command message currently being processed might not complete.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

1. The following command stops the command server on queue manager `saturn.queue.manager`:

   ```
   endmqcsv  -c saturn.queue.manager
   ```

   The command server can complete processing any command it has already started before it stops. Any new commands received remain unprocessed in the command queue until the command server is restarted.
2. The following command stops the command server on queue manager `pluto` immediately:

   ```
   endmqcsv -i pluto
   ```

Related commands

Command	Description
strmqcsv	Start a command server
dspmqcsv	Display the status of a command server

endmqlsr:

End all listener process for a queue manager.

Purpose

The **endmqlsr** command ends all listener processes for the specified queue manager.

You must use the **endmqlsr** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

You do not have to stop the queue manager before issuing the **endmqlsr** command. If any of the listeners are configured to have inbound channels running within the **runmqlsr** listener process, rather than within a pool process, the request to end that listener might fail if channels are still active. In this case a message is written indicating how many listeners were successfully ended and how many listeners are still running.

If the listener attribute, CONTROL, is specified as QMGR then changing the state of the listener using **endmqlsr** does not effect how the queue manager acts upon the CONTROL attribute at the next restart.

Syntax

Optional parameters

-m *QMgrName*
The name of the queue manager. If you omit this parameter, the command operates on the default queue manager.

-w Wait before returning control.

Control is returned to you only after all listeners for the specified queue manager have stopped.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

endmqdnm:

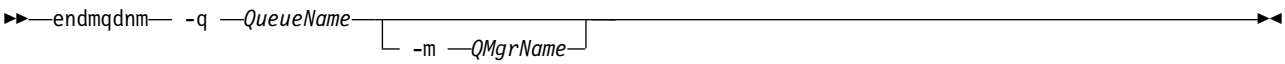
Stop the .NET monitor for a queue (Windows only).

Purpose

Note: The endmqdnm command applies to WebSphere MQ for Windows only.

Use the **endmqdnm** control command to stop a .NET monitor.

Syntax



Required parameters

-q *QueueName*
The name of the application queue that the .NET monitor is monitoring.

Optional parameters

-m *QMgrName*
The name of the queue manager that hosts the application queue.

If omitted, the default queue manager is used.

Return codes

Return code	Description
0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
58	Inconsistent use of installations detected
71	Unexpected error
72	Queue manager name error
133	Unknown object name error

endmqm:

Stop a queue manager or switch to a standby queue manager.

Purpose

Use the **endmqm** command to end (stop) a specified queue manager. This command stops a queue manager in one of three modes:

- Controlled or quiesced shutdown
- Immediate shutdown
- Pre-emptive shut down

The **endmqm** command stops all instances of a multi-instance queue manager in the same way as it stops a single instance queue manager. You can issue the **endmqm** on either the active instance, or one of the standby instances of a multi-instance queue manager. You must issue **endmqm** on the active instance to end the queue manager.

If you issue the **endmqm** command on the active instance of a multi-instance queue manager, you can permit a standby instance to switch over to being the new active instance when the current active instance completes its shutdown.

If you issue the **endmqm** command on a standby instance of a multi-instance queue manager, you can end the standby instance by adding the `-x` option, and leave the active instance running. The queue manager reports an error if you issue **endmqm** on the standby instance without the `-x` option.

Issuing the **endmqm** command will affect any client application connected through a server-connection channel. The effect varies depending on the parameter used, but it is as though a STOP CHANNEL command was issued in one of the three possible modes. See Stopping channels , for information about the effects of STOP CHANNEL modes on server-connection channels. The **endmqm** optional parameter descriptions state which STOP CHANNEL mode they will be equivalent to.

If you issue **endmqm** to stop a queue manager, reconnectable clients do not try to reconnect. To override this behavior, specify either the `-r` or `-s` option to enable clients to start trying to reconnect.

Note: If a queue manager or a channel ends unexpectedly, reconnectable clients start trying to reconnect.

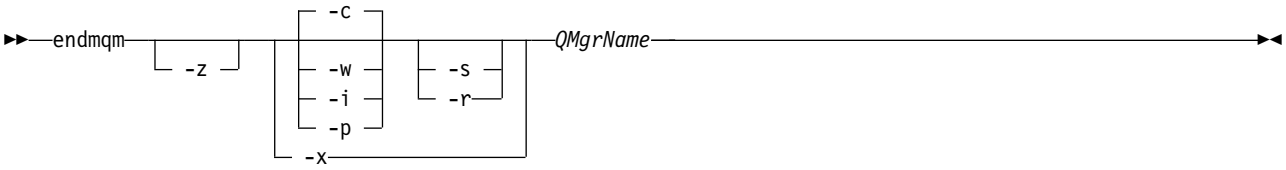
Note: The client might not reconnect to this queue manager. Depending on the MQCONNX reconnect option the client has used, and the definition of the queue manager group in the client connection table, the client might reconnect to a different queue manager. You can configure the client to force it to reconnect to the same queue manager.

You must use the **endmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

The attributes of the queue manager and the objects associated with it are not affected by the **endmqm** command. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, stop it and then use the **dltmqm** (Delete queue manager) command.

Syntax



Required parameters

QMgrName
The name of the message queue manager to be stopped.

Optional parameters

-c Controlled (or quiesced) shutdown. This parameter is the default.

The queue manager stops, but only after all applications have disconnected. Any MQI calls currently being processed are completed. In the unlikely event that a "dspmq" on page 163 command is issued in the small timeframe between the applications disconnecting and the queue manager actually stopping, the "dspmq" on page 163 command might transiently report the status as `Ending immediately`, even though a controlled shutdown was requested.

Control is returned to you immediately and you are not notified of when the queue manager has stopped.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in QUIESCE mode.

-i Immediate shutdown. The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI requests issued after the command has been issued fail. Any incomplete units of work are rolled back when the queue manager is next started.

Control is returned after the queue manager has ended.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in FORCE mode.

-p Pre-emptive shutdown.

Use this type of shutdown only in exceptional circumstances. For example, when a queue manager does not stop as a result of a normal **endmqm** command.

The queue manager might stop without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for WebSphere MQ applications. The shutdown mode is set to *immediate shutdown*. If the queue manager has not stopped after a few seconds, the shutdown mode is escalated, and all remaining queue manager processes are stopped.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in TERMINATE mode.

-r Start trying to reconnect reconnectable clients. This parameter has the effect of reestablishing the connectivity of clients to other queue managers in their queue manager group.

-s Switch over to a standby queue manager instance after shutting down. The command checks that there is a standby instance running before ending the active instance. It does not wait for the standby instance to start before ending.

Connections to the queue manager are broken by the active instance shutting down. Reconnectable clients start trying to reconnect.

You can configure the reconnection options of a client to reconnect only to another instance of the same queue manager, or to reconnect to other queue managers in the queue manager group.

-w Wait shutdown.

This type of shutdown is equivalent to a controlled shutdown except that control is returned to you only after the queue manager has stopped. You receive the message `Waiting for queue manager` *qmName* `to end` while shutdown progresses. In the unlikely event that a "dspmq" on page 163 command is issued in the small timeframe between the applications disconnecting and the queue manager actually stopping, the "dspmq" on page 163 command might transiently report the status as `Ending immediately`, even though a controlled shutdown was requested.

The effect on any client applications connected through a server-connection channel is equivalent to a STOP CHANNEL command issued in QUIESCE mode.

-x End a standby instance of the queue manager, without ending the active instance of the queue manager.

-z Suppresses error messages on the command.

Return codes

Return code	Description
0	Queue manager ended
3	Queue manager being created
16	Queue manager does not exist
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
62	The queue manager is associated with a different installation
69	Storage not available

Return code	Description
71	Unexpected error
72	Queue manager name error
77	WebSphere MQ queue manager cannot switch over
79	Active instance of WebSphere MQ queue manager *QmgrName* not ended
90	Standby instance of WebSphere MQ queue manager *QmgrName* not ended
119	Permission denied

Examples

The following examples show commands that stop the specified queue managers.

1. This command ends the queue manager named `mercury.queue.manager` in a controlled way. All applications currently connected are allowed to disconnect.

   ```
   endmqm mercury.queue.manager
   ```

2. This command ends the queue manager named `saturn.queue.manager` immediately. All current MQI calls complete, but no new ones are allowed.

   ```
   endmqm -i saturn.queue.manager
   ```

The results of issuing **endmqm** to the local instance of a multi-instance queue manager are shown in Table 47. The results of the command depend on whether the -s or -x switch is used, and the running status of local and remote instances of the queue manager.

Table 47. endmqm actions

endmqm option	Local machine	Remote machine	RC	Message	Result
	Active	None	0	-	Queue manager ended.
		Standby			Queue manager ended, including the standby instance.
	Standby	Active	90	AMQ8368	Standby instance of WebSphere MQ queue manager *QmgrName* not ended.
-s	Active	None	77	AMQ7276	WebSphere MQ queue manager cannot switch over.
		Standby	0	-	Queue manager QMNAME ended, permitting switchover to a standby instance.
	Standby	Active	90	AMQ8368	Standby instance of WebSphere MQ queue manager *QmgrName* not ended.
-x	Active	None	79	AMQ8367	Active instance of WebSphere MQ queue manager *QmgrName* not ended.
		Standby			
	Standby	Active	0	-	Standby instance of queue manager QMNAME ended.

Related commands

Command	Description
"**crtmqm**" on page 146	Create queue manager
"strmqm" on page 243	Start queue manager
"dltmqm" on page 153	Delete queue manager

endmqsvc (end IBM IBM WebSphere MQ service):

The **endmqsvc** command ends the IBM IBM WebSphere MQ service on Windows. Run the command on Windows only.

Purpose

The command ends the IBM IBM WebSphere MQ service on Windows.

Run the command to end the service, if the service is running.

Restart the service for IBM WebSphere MQ processes to pick up a new environment, including new security definitions.

Syntax

endmqsvc

Parameters

The **endmqsvc** command has no parameters.

You must set the path to the installation that contains the service. Either make the installation primary, run the **setmqenv** command, or run the command from the directory containing the **endmqsvc** binary file.

Related reference:

"strmqsvc (Start IBM IBM WebSphere MQ service)" on page 242
The **strmqsvc** command starts the IBM IBM WebSphere MQ service on Windows. Run the command on Windows only.

endmqtrc:

End trace for some or all of the entities that are being traced.

Purpose

Use the **endmqtrc** command to end tracing for the specified entity or all entities. The **endmqtrc** command ends only the trace that is described by its parameters. Using **endmqtrc** with no parameters ends early tracing of all processes.

Attention: There can be a slight delay between the **endmqtrc** command ending, and all trace operations actually completing. This is because WebSphere MQ processes are accessing their own trace files. As each process becomes active at different times, their trace files close independently of one another.

Syntax

The syntax of this command is as follows:

Optional parameters

-m *QMgrName*
The name of the queue manager for which to end tracing. This parameter applies to server products only.

The QMgrName supplied must match exactly the QMgrName supplied on the strmqtrc command. If the strmqtrc command used wildcards, the endmqtrc command must use the same wildcard specification including the escaping of any wildcard characters to prevent them being processed by the command environment.

A maximum of one -m flag and associated queue manager name can be supplied on the command.

-i *PidTids*
Process identifier (PID) and thread identifier (TID) for which to end tracing. You cannot use the -i flag with the -e flag. If you try to use the -i flag with the -e flag, then an error message is issued. This parameter must only be used under the guidance of IBM Service personnel.

-p *Apps*
The named processes for which to end tracing. *Apps* is a comma-separated list. You must specify each name in the list exactly as the program name would be displayed in the "Program Name" FDC header. Asterisk (*) or question mark (?) wildcards are allowed. You cannot use the -p flag with the -e flag. If you try to use the -p flag with the -e flag, then an error message is issued.

-e Ends early tracing of all processes.

Using endmqtrc with no parameters has the same effect as endmqtrc -e. You cannot specify the -e flag with the -m flag, the -i flag, or the -p flag.

-a Ends all tracing.

This flag ***must*** be specified alone.

Return codes

Return code	Description
AMQ5611	This message is issued if you supply invalid arguments to the command.
58	Inconsistent use of installations detected

Examples

This command ends tracing of data for a queue manager called QM1.

```
endmqtrc -m QM1
```

The following examples are a sequence that shows how the **endmqtrc** command ends only the trace that is described by its parameters.

1. The following command enables tracing for queue manager QM1 and process amqxxx.exe:
   ```
   strmqtrc -m QM1 -p amqxxx.exe
   ```
2. The following command enables tracing for queue manager QM2:
   ```
   strmqtrc -m QM2
   ```
3. The following command ends tracing for queue manager QM2 only. Tracing of queue manager QM1 and process amqxxx.exe continues:
   ```
   endmqtrc -m QM2
   ```

Related commands

Command	Description
dspmqtrc	Display formatted trace output
“strmqtrc” on page 247	Start trace

migmbbrk:

The migmbbrk command migrates publish/subscribe configuration data from WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 to WebSphere MQ Version 7.0.1 or later versions.

Purpose

The migmbbrk command is not supported on all of the platforms that WebSphere MQ supports. See *Supported operating systems* **for details.**

To use the **migmbbrk** command you must be using at least WebSphere Message Broker Version 6.0, Fix Pack 9, or WebSphere Message Broker Version 6.1, Fix Pack 4.

Use the **migmbbrk** command to migrate the publish/subscribe configuration data from a WebSphere Event Broker Version 6.0 or a WebSphere Message Broker Version 6.0 or Version 6.1 broker to a WebSphere MQ Version 7.0.1 or later queue manager. The command runs a migration process that migrates the following publish/subscribe configuration data to the queue manager that is associated with the named broker:

- Subscriptions
- Subscription points. (Subscription points are supported only when RFH2 messages are used.)
- Streams
- Retained publications

The **migmbbrk** command does not migrate the Access Control List (ACL). Instead, running the migration with the -t or -r parameters produces a file containing suggested setmqaut commands to set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. You must review and modify the security command file as needed and run the commands to set up a security environment in the queue manager, equivalent to the one that existed in the broker, before you run the migration with the -c parameter to complete the migration.

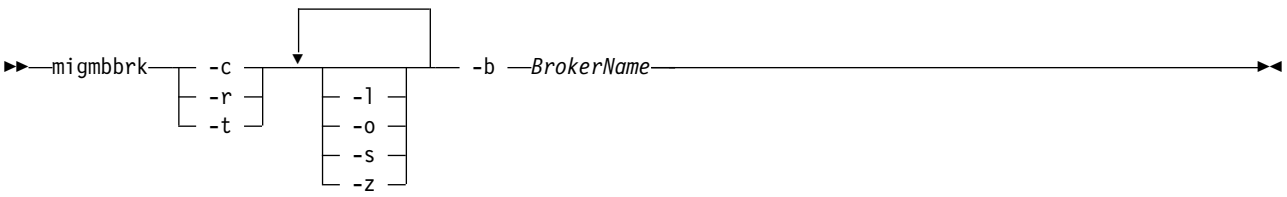
Note: On UNIX systems, all authorities are held by user groups internally, not by principals. This has the following implications:

- If you use the **setmqaut** command to grant an authority to a principal, the authority is granted to the primary user group of the principal. This means that the authority is effectively granted to all members of that user group.
- If you use the **setmqaut** command to revoke an authority from a principal, the authority is revoked from the primary user group of the principal. This means that the authority is effectively revoked from all members of that user group.

You must issue the **migmbbrk** command from a command window that can execute both WebSphere MQ and WebSphere Message Broker commands successfully. Typically this is true if the command is issued from a WebSphere Message Broker command console.

The WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0 or 6.1 publish/subscribe configuration data, which is stored in the subscription database tables, is not deleted by the migration process. This configuration data is therefore available to use until you explicitly delete it.

Syntax



Required parameters

-b *BrokerName*
The name of the broker that is the source of the publish/subscribe configuration data that is to be migrated. The queue manager to which the publish/subscribe configuration data is migrated is the queue manager that is associated with the named broker.

-c
Complete the migration of the publish/subscribe configuration data. The completion phase of the migration uses the topic objects that are created in the initial -t phase. It is possible that the broker state has changed since the initial phase was run and that new additional topic objects are now required. If so, the completion phase creates new topic objects as necessary. The completion phase does not delete any topic objects that have become unnecessary; you might need to delete any topic objects that you do not require.

Before you complete the migration you must review and modify the security command file produced in the -r or -t phase as required and execute the commands to set up a security environment in the queue manager, equivalent to the one that existed in the broker.

Before you run this completion phase, you must run the initial -t phase. You cannot use the -c parameter with the -r parameter or the -t parameter. This phase also creates a migration log.

-r
Rehearse the migration process but do not change anything. You can use this parameter before running the migration with the -t parameter, to create a migration log, including any errors, so that you can observe what the result of the migration process would be, but without changing the current configurations.

Rehearsing the migration also produces a file containing suggested setmqaut commands to set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. Before you complete the migration with the -c parameter you must review and modify the security command file as required and execute the commands to set up a security environment in the queue manager, equivalent to the one that existed in the broker.

You cannot use the -r parameter with the -c parameter or the -t parameter.

-t
Create topic objects that might be needed in the queue manager, based on the ACL entries that are defined in the broker.

Use of the -t parameter also produces a file containing suggested setmqaut commands to set up a security environment in the queue manager that is equivalent to the security environment that existed in the broker. The topic objects are created in anticipation of you executing the security commands to create ACLs for the topic objects. Before you complete the migration with the -c parameter you must review and modify the security command file as required and execute the commands to set up a security environment in the queue manager, equivalent to the one that existed in the broker.

You must run this phase before you run the completion phase with the -c parameter. You cannot use the -t parameter with the -c parameter or the -r parameter. This phase also creates a migration log.

Optional parameters

-l
Leave the broker running. If you do not specify this parameter, the broker is shut down by default at the end of the migration process.

-o
Overwrite any subscription or retained publication that exists in the queue manager and that has the same name as a subscription or retained publication that is being migrated from the broker, with the publish/subscribe configuration data that was retrieved from the broker. The -o parameter has no effect if you use it with the -r parameter.

-s
Discard any intermediate configuration data that was retained from a previous instance of the migration process that failed or was interrupted. The migration process populates private queues with temporary data. If the migration process completes successfully, the temporary data is deleted. If you do not specify this parameter and the migration process fails or is interrupted, the temporary data is retained and is used by the migration process if you restart it, so that the process resumes at the point where it previously failed or was interrupted.

-z
Run the migration process, regardless of whether it has previously run to a successful completion. If you do not specify this parameter and the migration process has previously run to a successful completion, the process recognizes this fact and exits. You can use the -o parameter with the -z parameter, but this is not mandatory. A previous rehearsal of the migration using the -r parameter does not count as a successful completion.

Return codes

Return Code	Explanation
0	Migration completed successfully
20	An error occurred during processing

Output files

The migration process writes two output files to the current directory:

amqmigrateacl.txt
A file containing a list of setmqaut commands, created in the current directory for you to review, change, and run if appropriate, to help you to reproduce your ACLs.

amqmigmbbrk.log
A log file containing a record of the details of the migration.

Examples

This command migrates the publish/subscribe configuration data of broker BRK1 into its associated queue manager and specifies that the migration process runs regardless of whether it has previously run to a successful completion. It also specifies that any subscription or retained publication that exists in the queue manager, that has the same name as a subscription or retained publication that is being migrated from the broker, must be overwritten.

```
migmbbrk -z -o -b BRK1
```

Supported operating systems

The **migmbbrk** command is supported only on the following platforms that support WebSphere Event Broker Version 6.0 or WebSphere Message Broker Version 6.0:

Microsoft Windows XP Professional with SP2, 32-bit versions only
Solaris x86-64 platform: Solaris 10
Solaris SPARC platform: Sun Solaris 9 (64-bit)
AIX Version 5.2 or later, 64-bit only
HP-UX Itanium platform: HP-UX 11i
Linux zSeries (64-bit)
Linux PowerPC® (64-bit)
Linux Intel x86
Linux Intel x86-64

On z/OS, the equivalent function to the migmbbrk command is provided by the CSQUMGMB utility.

MQExplorer (launch WebSphere MQ Explorer):

Start IBM WebSphere MQ Explorer (Windows, Linux x86, and Linux x86-64 platforms only).

Purpose

To launch IBM WebSphere MQ Explorer by using the system menu on Linux, or the start menu on Windows, you must left-click on the installation that you want to launch.

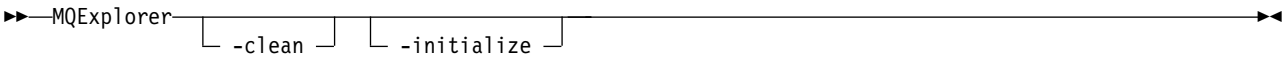
On Windows, open the start menu, and select the IBM WebSphere MQ Explorer installation entry under the **IBM WebSphere MQ** folder that corresponds to the installation that you want to launch. Each instance of IBM WebSphere MQ Explorer listed is identified by the name that you chose for its installation.

On Linux, the system menu entry for IBM WebSphere MQ Explorer is added to the **Development** category. Where it appears within the system menu is dependent on your Linux distribution (SUSE or Red Hat), and your desktop environment (GNOME or KDE).

- On SUSE
  - Left-click **Computer > More Applications...,** and find the installation of IBM WebSphere MQ Explorer that you want to launch under the **Development** category.
- On Red Hat
  - The installation of IBM WebSphere MQ Explorer that you want to launch can be found under **Applications > Programming**.

Syntax

The **MQExplorer** command is stored in `MQ_INSTALLATION_PATH/bin`. **MQExplorer.exe** (the MQExplorer command) supports standard Eclipse runtime options. The syntax of this command is as follows:



Optional parameters

-clean
Is passed to Eclipse. This parameter causes Eclipse to delete any cached data used by the Eclipse runtime.

-initialize
Is passed to Eclipse. This parameter causes Eclipse to discard configuration information used by the Eclipse runtime.

The graphical user interface (GUI) does not start.

mqrc (MQ return code):

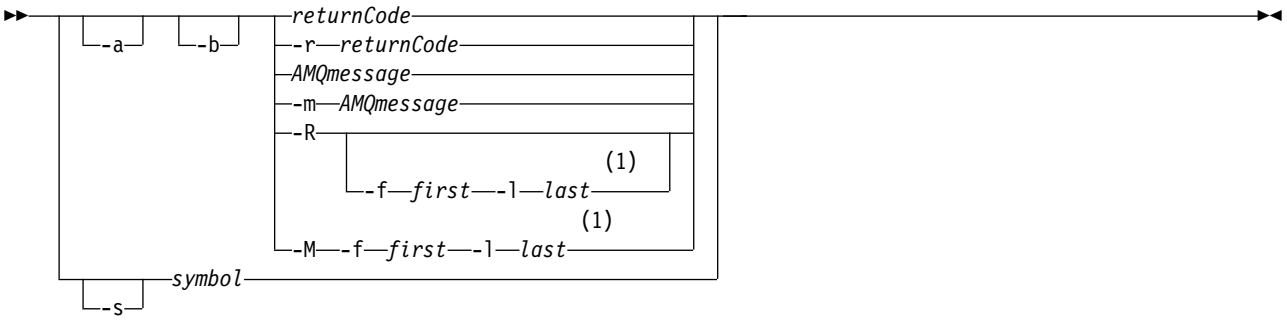
Display information about return codes.

Purpose

You can use the **mqrc** command to display information about symbols, return codes, and AMQ messages. You can specify a range of return codes or AMQ messages, as well as specifying specific return codes or AMQ messages.

Numeric arguments are interpreted as decimal if they start with a digit 1 - 9, or hex if prefixed with 0x.

Syntax



Notes:

1 If there is a problem with a message within a range, an indication is displayed before the message text. ? is displayed if there are no matching return codes for the message. ! is displayed if the message severity is different to the return code severity.

Parameters

returnCode
The return code to display

AMQmessage
The AMQ message to display

symbol
The symbol to display

-a Try all severities to find message text

-b Display messages without extended information

-f *first*
First number in a range

-l *last*
Last number in a range

-m *AMQmessage*
The AMQ message to list

-M Display AMQ messages in a range

-r *returnCode*
The return code to display

-R Display all return codes. If used with the **-f** and **-l** parameters, **-R** displays the return codes within a range.

-s *symbol*
The symbol to display

Examples

1. This command displays AMQ message 5005:

```
mqrc AMQ5005
```

2. This command displays return codes in the range 2505 - 2530:

```
mqrc -R -f 2505 -l 2530
```

rcdmqimg:

Write the image of an object or group of objects to the log for media recovery.

Purpose

Use the **rcdmqimg** command to write an image of an object, or group of objects, to the log for use in media recovery. This command can be used only when using linear logging. See Types of logging for more information about linear logging. Use the associated command **rcrmqobj** to recreate the object from the image.

rcdmqimg must be run manually or from an automated task you have created. The command does not run automatically as it must be run in accordance with, and as determined by, the usage of each individual customer of WebSphere MQ .

Running **rcdmqimg** moves the log sequence number (LSN) forwards and frees up old log files for archival or deletion.

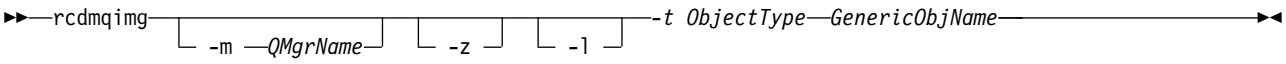
When determining when and how often to run **rcdmqimg**, consider these factors:

Disk space
If disk space is limited, regular running of **rcdmqimg** releases log files for archive or deletion.

Impact on normal system performance
rcdmqimg activity can take a long time if the queues on the system are deep. At this time, other system usage is slower and disk utilization increases because data is being copied from the queue files to the logs. Therefore, the ideal time to run **rcdmqimg** is when the queues are empty and the system is not being heavily used.

You use this command with an active queue manager. Further activity on the queue manager is logged so that, although the image becomes out of date, the log records reflect any changes to the object.

Syntax



Required parameters

GenericObjName
The name of the object to record. This parameter can have a trailing asterisk to record that any objects with names matching the portion of the name before the asterisk.

This parameter is required unless you are recording a queue manager object or the channel synchronization file. Any object name you specify for the channel synchronization file is ignored.

-t *ObjectType*
The types of object for which to record images. Valid object types are:

all and *	All the object types; **ALL** for objtype and * for GenericObjName
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or **chl**	Channels
clntconn or **clcn**	Client connection channels
catalog or **ctlg**	An object catalog
listener or **lstr**	Listeners
namelist or **nl**	Namelists
process or **prcs**	Processes
queue or **q**	All types of queue
qalias or **qa**	Alias queues
qlocal or **ql**	Local queues
qmodel or **qm**	Model queues
qremote or **qr**	Remote queues
qmgr	Queue manager object
service or **srvc**	Service
syncfile	Channel synchronization file.
topic or **top**	Topics

Note: When using IBM WebSphere MQ for UNIX systems, you must prevent the shell from interpreting the meaning of special characters, for example, an asterisk (*). How you do this depends on the shell you are using, but might involve the use of single quotation marks ('), double quotation marks ("), or a backslash (\).

Optional parameters

-m *QMgrName*
The name of the queue manager for which to record images. If you omit this parameter, the command operates on the default queue manager.

-z Suppresses error messages.

-l Writes messages containing the names of the oldest log files required to restart the queue manager and to perform media recovery. The messages are written to the error log and the standard error destination. (If you specify both the -z and -l parameters, the messages are sent to the error log, but not to the standard error destination.)

When issuing a sequence of **rcdmqimg** commands, include the -l parameter only on the last command in the sequence, so that the log file information is gathered only once.

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
68	Media recovery not supported
69	Storage not available
71	Unexpected error

Return code	Description
72	Queue manager name error
119	User not authorized
128	No objects processed
131	Resource problem
132	Object damaged
135	Temporary object cannot be recorded

Examples

The following command records an image of the queue manager object `saturn.queue.manager` in the log.

```
rcdmqimg -t qmgr -m saturn.queue.manager
```

Related commands

Command	Description
rcrmqobj	Recreate a queue manager object

rcrmqobj:

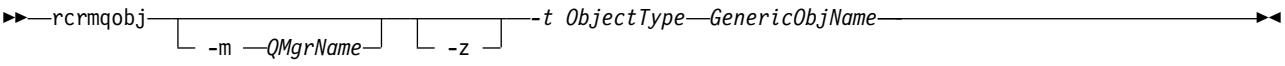
Re-create an object, or group of objects, from their images contained in the log.

Purpose

Use this command to re-create an object, or group of objects, from their images contained in the log. This command can only be used when using linear logging. Use the associated command, **rcdmqimg**, to record the object images to the log.

Use this command on a running queue manager. All activity on the queue manager after the image was recorded is logged. To re-create an object, replay the log to re-create events that occurred after the object image was captured.

Syntax



Required parameters

GenericObjName
The name of the object to re-create. This parameter can have a trailing asterisk to re-create any objects with names matching the portion of the name before the asterisk.

This parameter is required ***unless*** the object type is the channel synchronization file; any object name supplied for this object type is ignored.

-t ObjectType
The types of object to re-create. Valid object types are:

* or **all**	All object types
authinfo	Authentication information object, for use with Secure Sockets Layer (SSL) channel security
channel or **chl**	Channels
clntconn or **clcn**	Client connection channels
clchltab	Client channel table
listener or **lstr**	Listener
namelist or **nl**	Namelists
process or **prcs**	Processes
queue or **q**	All types of queue
qalias or **qa**	Alias queues
qlocal or **ql**	Local queues
qmodel or **qm**	Model queues
qremote or **qr**	Remote queues
service or **srvc**	Service
syncfile	Channel synchronization file. You can use this option when circular logs are configured but the `syncfile` fails if the channel scratchpad files, which are used to rebuild `syncfile`, are damaged or missing. You might want to do this if your system has reported the error message AMQ7353 (krcE_SYNCFILE_UPDATE_FAILED).
topic or **top**	Topics

Note: When using WebSphere MQ for UNIX systems, you must prevent the shell from interpreting the meaning of special characters, for example, an asterisk (*). How you do this depends on the shell you are using, but might involve the use of single quotation marks ('), double quotation marks ("), or a backslash (\).

Optional parameters

-m *QMgrName*
The name of the queue manager for which to re-create objects. If omitted, the command operates on the default queue manager.

-z Suppresses error messages.

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
66	Media image not available
68	Media recovery not supported
69	Storage not available
71	Unexpected error
72	Queue manager name error
119	User not authorized

Return code	Description
128	No objects processed
135	Temporary object cannot be recovered
136	Object in use

Examples

1. The following command re-creates all local queues for the default queue manager:

   ```
   rcrmqobj -t ql *
   ```

2. The following command re-creates all remote queues associated with queue manager `store`:

   ```
   rcrmqobj -m store -t qr *
   ```

Related commands

Command	Description
rcdmqimg	Record an object in the log

rmvmqinf:

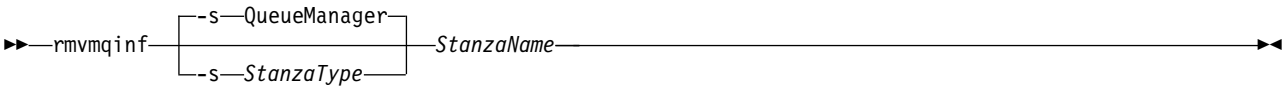
Remove WebSphere MQ configuration information (Windows and UNIX platforms only).

Purpose

Use the **rmvmqinf** command to remove WebSphere MQ configuration information.

You must use the **rmvmqinf** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

Syntax



Required parameters

StanzaName
The name of the stanza. That is, the value of the key attribute that distinguishes between multiple stanzas of the same type.

Optional parameters

-s *StanzaType*
The type of stanza to remove. If omitted, a QueueManager stanza is removed.

The only supported value of *StanzaType* is `QueueManager`.

Return codes

Return code	Description
0	Successful operation
5	Queue manager is running
26	Queue manager is running as a standby instance
39	Bad command line parameters
44	Stanza does not exist
49	Queue manager is stopping
58	Inconsistent use of installations detected
69	Storage is not available
71	Unexpected error
72	Queue manager name error

Example

```
rmvmqinf QM.NAME
```

Usage notes

Use **rmvmqinf** to remove an instance of a multi-instance queue manager.

To use this command you must be a WebSphere MQ administrator and a member of the `mqm` group.

Related commands

Command	Description
"addmqinf" on page 133	Add queue manager configuration information
"dspmqinf" on page 172	Display queue manager configuration information

rsvmqtrn:

Resolve in-doubt and heuristically completed transactions

Purpose

The **rsvmqtrn** command is used to resolve two different transaction states.

in-doubt transactions
: Use the **rsvmqtrn** command to commit or back out internally or externally coordinated in-doubt transactions.

  Note: Use this command only when you are certain that transactions cannot be resolved by the normal protocols. Issuing this command might result in the loss of transactional integrity between resource managers for a distributed transaction.

heuristically completed transactions
: Use the **rsvmqtrn** command with the -f option for WebSphere MQ to remove all information about externally coordinated transactions that were previously resolved manually using the **rsvmqtrn** command, but the resolution has not been acknowledged by the transaction coordinator using the xa-forget command. Transactions that are manually resolved by a resource manager and unacknowledged by the transaction manager, are known as *heuristically completed* transactions by X/Open.

  Note: Only use the -f option if the external transaction coordinator is permanently unavailable. The queue manager, as a resource manager, remembers the transactions that are committed or backed out manually by the **rsvmqtrn** command.

Syntax

Required parameters

-m *QMgrName*
The name of the queue manager.

Optional parameters

-a The queue manager resolves all internally coordinated, in-doubt transactions (that is, all global units of work).

-b Backs out the named transaction. This flag is valid for externally coordinated transactions (that is, for external units of work) only.

-c Commits the named transaction. This flag is valid for externally coordinated transactions (that is, external units of work) only.

-f
Forgets the named heuristically completed transaction. This flag is valid only for externally coordinated transactions (that is, external units of work) that are resolved, but unacknowledged by the transaction coordinator.

Note: Use only if the external transaction coordinator is never going to be able to acknowledge the heuristically completed transaction. For example, if the transaction coordinator has been deleted.

-r *RMID*
The participation of the resource manager in the in-doubt transaction can be ignored. This flag is valid for internally coordinated transactions only, and for resource managers that have had their resource manager configuration entries removed from the queue manager configuration information.

Note: The queue manager does not call the resource manager. Instead, it marks the participation of the resource manager in the transaction as being complete.

Transaction
The transaction number of the transaction being committed or backed out. Use the **dspmqtrn** command to find the relevant transaction number. This parameter is required with the -b, -c, and -r *RMID* parameters, and if used it must be the last parameter.

Return codes

Return code	Description
0	Successful operation
26	Queue manager running as a standby instance.
32	Transactions could not be resolved
34	Resource manager not recognized
35	Resource manager not permanently unavailable
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error

Return code	Description
85	Transactions not known

Related commands

Command	Description
dspmqtrn	Display list of prepared transactions

runmqchi:

Run a channel initiator process to automate starting channels.

Purpose

Use the **runmqchi** command to run a channel initiator process.

You must use the **runmqchi** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

The channel initiator is started by default as part of the queue manager.

Syntax

Optional parameters

-q *InitiationQName*
The name of the initiation queue to be processed by this channel initiator. If you omit it, SYSTEM.CHANNEL.INITQ is used.

-m *QMgrName*
The name of the queue manager on which the initiation queue exists. If you omit the name, the default queue manager is used.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

If errors occur that result in return codes of either 10 or 20, review the queue manager error log that the channel is associated with for the error messages, and the system error log for records of problems that occur before the channel is associated with the queue manager. For more information about error logs, see Error log directories.

runmqchl:

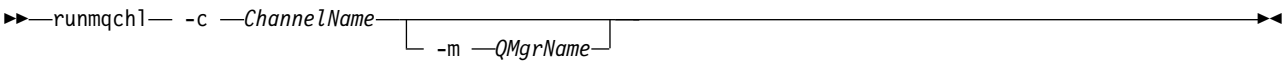
Start a sender or requester channel

Purpose

Use the **runmqchl** command to run either a sender (SDR) or a requester (RQSTR) channel.

The channel runs synchronously. To stop the channel, issue the MQSC command STOP CHANNEL.

Syntax



Required parameters

-c *ChannelName*
The name of the channel to run.

Optional parameters

-m *QMgrName*
The name of the queue manager with which this channel is associated. If you omit the name, the default queue manager is used.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages, and the system error log for records of problems that occur before the channel is associated with the queue manager.

runmqdlq:

Start the dead-letter queue handler to monitor and process messages on the dead-letter queue.

Purpose

Use the **runmqdlq** command to start the dead-letter queue (DLQ) handler, which monitors and handles messages on a dead-letter queue.

Syntax

Description

Use the dead-letter queue handler to perform various actions on selected messages by specifying a set of rules that can both select a message and define the action to be performed on that message.

The **runmqdlq** command takes its input from `stdin`. When the command is processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter **runmqdlq** rules interactively.

By redirecting the input from a file, you can apply a rules table to the specified queue. The rules table must contain at least one rule.

If you use the DLQ handler without redirecting stdin from a file (the rules table), the DLQ handler reads its input from the keyboard. In WebSphere MQ for AIX, Solaris, HP-UX, and Linux, the DLQ handler does not start to process the named queue until it receives an end_of_file (Ctrl+D) character. In WebSphere MQ for Windows, it does not start to process the named queue until you press the following sequence of keys: Ctrl+Z, Enter, Ctrl+Z, Enter.

For more information about rules tables and how to construct them, see The DLQ handler rules table.

Optional parameters

The MQSC command rules for comment lines and for joining lines also apply to the DLQ handler input parameters.

QName
The name of the queue to be processed.

If you omit the name, the dead-letter queue defined for the local queue manager is used. If you enter one or more blanks (' '), the dead-letter queue of the local queue manager is explicitly assigned.

QMgrName
The name of the queue manager that owns the queue to be processed.

If you omit the name, the default queue manager for the installation is used. If you enter one or more blanks (' '), the default queue manager for this installation is explicitly assigned.

runmqdnm:

Start processing messages on a queue using the .NET monitor (Windows only).

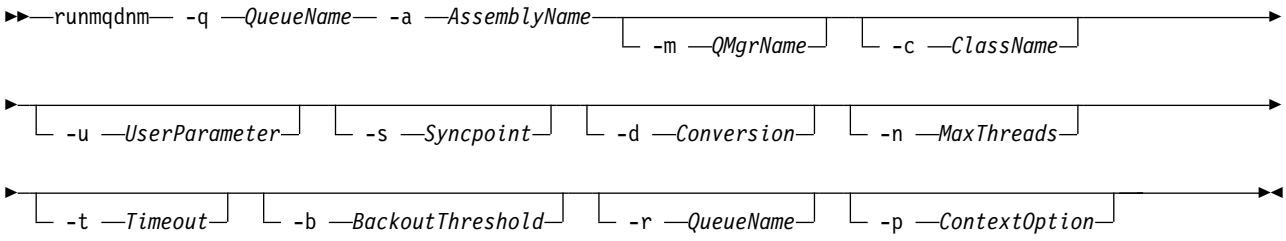
Purpose

Note: The runmqdnm command applies to WebSphere MQ for Windows only.

runmqdnm can be run from the command line, or as a triggered application.

Use the **runmqdnm** control command to start processing messages on an application queue with a .NET monitor.

Syntax



Required parameters

-q *QueueName*
The name of the application queue to monitor.

-a *AssemblyName*
The name of the .NET assembly.

Optional parameters

-m *QMgrName*
The name of the queue manager that hosts the application queue.

If omitted, the default queue manager is used.

-c *ClassName*
The name of the .NET class that implements the IMQObjectTrigger interface. This class must reside in the specified assembly.

If omitted, the specified assembly is searched to identify classes that implement the IMQObjectTrigger interface:

- If one class is found, then *ClassName* takes the name of this class.
- If no classes or multiple classes are found, then the .NET monitor is not started and a message is written to the console.

-u *UserData*
User-defined data. This data is passed to the Execute method when the .NET monitor calls it. User data must contain ASCII characters only, with no double quotation marks, NULLs, or carriage returns.

If omitted, null is passed to the Execute method.

-s *Syncpoint*
Specifies whether sync point control is required when messages are retrieved from the application queue. Possible values are:

YES	Messages are retrieved under sync point control (MQGMO_SYNCPOINT).
NO	Messages are not retrieved under sync point control (MQGMO_NO_SYNCPOINT).
PERSISTENT	Persistent messages are retrieved under sync point control (MQGMO_SYNCPOINT_IF_PERSISTENT).

If omitted, the value of *Syncpoint* is dependent on your transactional model:

- If distributed transaction coordination (DTC) is being used, then *Syncpoint* is specified as YES.
- If distributed transaction coordination (DTC) is not being used, then *Syncpoint* is specified as PERSISTENT.

-d *Conversion*
Specifies whether data conversion is required when messages are retrieved from the application queue. Possible values are:

YES	Data conversion is required (MQGMO_CONVERT).
NO	Data conversion is not required (no get message option specified).

If omitted, *Conversion* is specified as NO.

-n *MaxThreads*
The maximum number of active worker threads.

If omitted, *MaxThreads* is specified as 20.

-t *Timeout*
The time, in seconds, that the .NET monitor waits for further messages to arrive on the application queue. If you specify -1, the .NET monitor waits indefinitely.

If omitted when run from the command line, the .NET monitor waits indefinitely.

If omitted when run as a triggered application, the .NET monitor waits for 10 seconds.

-b *BackoutThreshold*
Specifies the backout threshold for messages retrieved from the application queue. Possible values are:

-1	The backout threshold is taken from the application queue attribute, BOTHRESH.
0	The backout threshold is not set.
1 or more	Explicitly sets the backout threshold.

If omitted, *BackoutThreshold* is specified as -1.

-r *QueueName*
The queue to which messages, with a backout count exceeding the backout threshold, are put.

If omitted, the value of *QueueName* is dependent on the value of the BOQNAME attribute from the application queue:

- If BOQNAME is non-blank, then *QueueName* takes the value of BOQNAME.
- If BOQNAME is blank, then *QueueName* is specified as the queue manager dead letter queue. If a dead letter queue has not been assigned to the queue manager, then backout processing is not available.

-p *ContextOption*
Specifies whether context information from a message that is being backed out is passed to the backed out message. Possible values are:

NONE	No context information is passed.
IDENTITY	Identity context information is passed only.
ALL	All context information is passed.

If omitted, *ContextOption* is specified as ALL.

Return codes

Return code	Description
0	Successful operation
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
71	Unexpected error
72	Queue manager name error
133	Unknown object name error

runmqlsr:

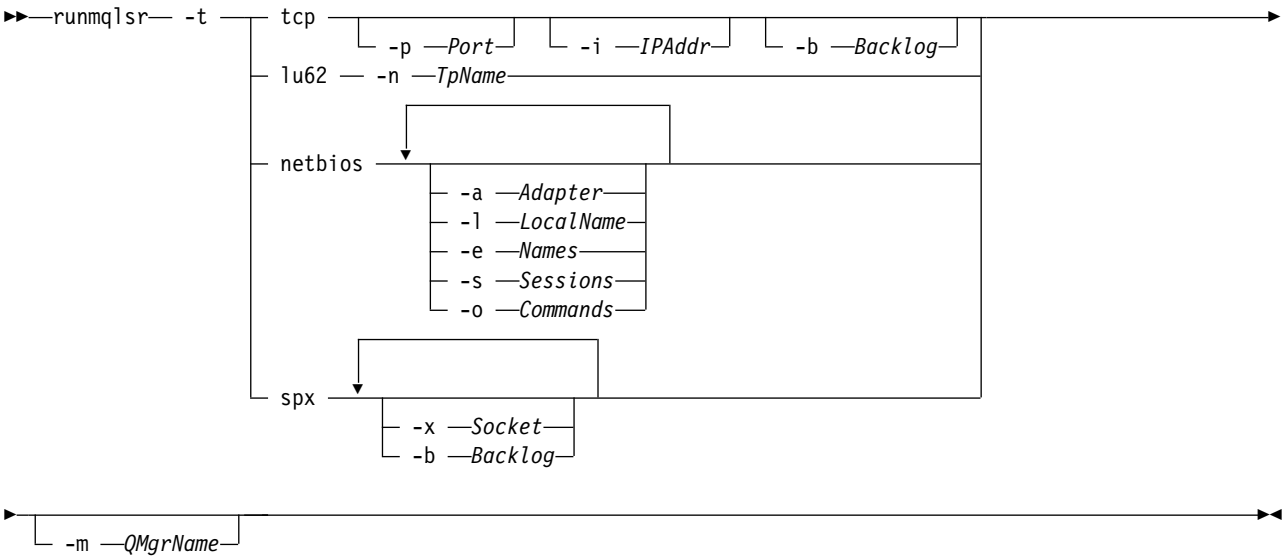
Run a listener process to listen for remote requests on various communication protocols.

Purpose

Use the **runmqlsr** command to start a listener process.

This command is run synchronously and waits until the listener process has finished before returning to the caller.

Syntax



Required parameters

-t The transmission protocol to be used:

`tcp`	Transmission Control Protocol / Internet Protocol (TCP/IP)
`lu62`	SNA LU 6.2 (Windows only)
`netbios`	NetBIOS (Windows only)
`spx`	SPX (Windows only)

Optional parameters

-p *Port*
The port number for TCP/IP. This flag is valid for TCP only. If you omit the port number, it is taken from the queue manager configuration information, or from defaults in the program. The default value is 1414. It must not exceed 65535.

-i *IPAddr*
The IP address for the listener, specified in one of the following formats:

- IPv4 dotted decimal
- IPv6 hexadecimal notation
- Alphanumeric format

This flag is valid for TCP/IP only.

On systems that are both IPv4 and IPv6 capable you can split the traffic by running two separate listeners. One listening on all IPv4 addresses and one listening on all IPv6 addresses. If you omit this parameter, the listener listens on all configured IPv4 and IPv6 addresses.

-n *TpName*
The LU 6.2 transaction program name. This flag is valid only for the LU 6.2 transmission protocol. If you omit the name, it is taken from the queue manager configuration information.

-a *Adapter*
The adapter number on which NetBIOS listens. By default the listener uses adapter 0.

-l *LocalName*
The NetBIOS local name that the listener uses. The default is specified in the queue manager configuration information.

-e *Names*
The number of names that the listener can use. The default value is specified in the queue manager configuration information.

-s *Sessions*
The number of sessions that the listener can use. The default value is specified in the queue manager configuration information.

-o *Commands*
The number of commands that the listener can use. The default value is specified in the queue manager configuration information.

-x *Socket*
The SPX socket on which SPX listens. The default value is hexadecimal 5E86.

-m *QMgrName*
The name of the queue manager. By default the command operates on the default queue manager.

-b *Backlog*
The number of concurrent connection requests that the listener supports. See TCP, LU62, NETBIOS, and SPX for a list of default values and further information.

Return codes

Return code	Description
0	Command completed normally
4	Command completed after being ended by the **endmqlsr** command
10	Command completed with unexpected results
20	An error occurred during processing: the AMQMSRVN process did not start.

Examples

The following command runs a listener on the default queue manager using the NetBIOS protocol. The listener can use a maximum of five names, five commands, and five sessions. These resources must be within the limits set in the queue manager configuration information.

```
runmqlsr -t netbios -e 5 -s 5 -o 5
```

runmqras:

Use the **runmqras** command to gather IBM WebSphere MQ diagnostic information together into a single archive, for example to submit to IBM Support.

Purpose

The **runmqras** command is used to gather diagnostic information from a machine, into a single archive. You can use this command to gather information about an application or IBM WebSphere MQ failure, possibly for submitting to IBM when you report a problem.

By default, **runmqras** gathers information such as:

- IBM WebSphere MQ FDC files
- Error logs (from all queue managers as well as the machine-wide IBM WebSphere MQ error logs)
- Product versioning, status information, and output from various other operating system commands.

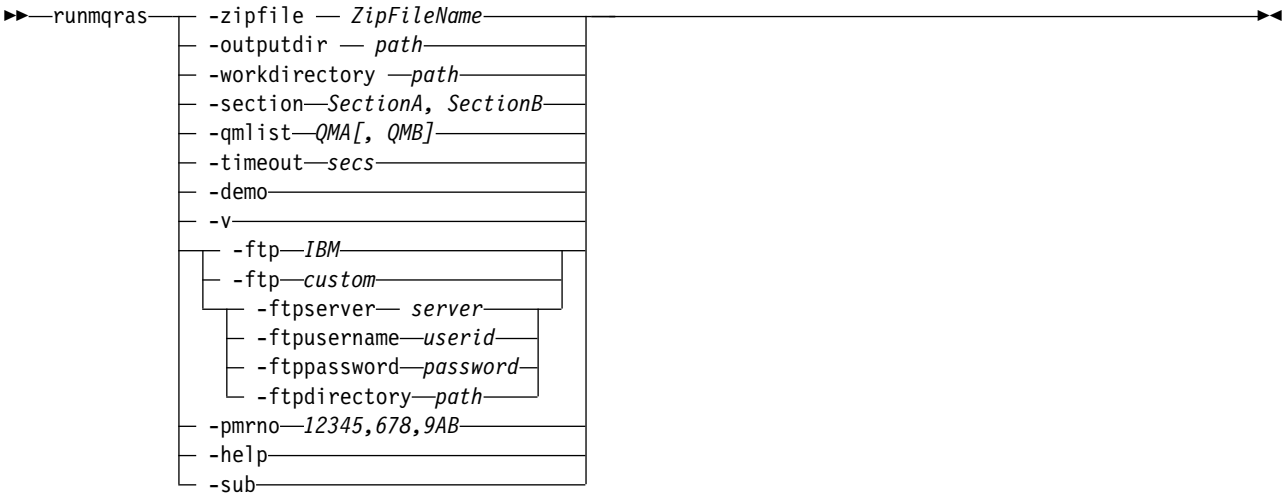
  Note, for example, the **runmqras** command does not gather user information that is contained in messages on queues.

Running without requesting more sections is intended as a starting point for general problem diagnosis, however, you can request more *sections* through the command line.

These additional *sections* gather more detailed information, depending on the type of problem being diagnosed. If non-default sections are needed by IBM support personnel, they will tell you.

The **runmqras** command can be run under any user ID, but the command only gathers information that the user ID can gather manually. In general, when debugging IBM WebSphere MQ problems, run the command under the `mqm` user ID to allow the command to gather queue manager files and command outputs.

Syntax



Keywords and parameters

All parameters are required unless the description states they are optional.

In every case, *QMgrName* is the name of the queue manager to which the command applies.

-zipfile *ZipFileName*
Supply the file name of the resulting archive.

By default, the name of the output archive is `runmqras.zip`.

-outputdir *path*
The directory in which the resulting output file is placed.

By default, the output directory is the same as the work directory.

-workdirectory *path*
The directory that is used for storing the output from commands that are run during the processing of the tool. If supplied, this directory must either not exist, in which case it is created, or must be empty.

If you do not supply the path, a directory under `/tmp` is used on UNIX systems, and under `%temp%` is used on Windows, whose name starts with **runmqras** and is suffixed by the date and time.

-section *SectionA*, *SectionB*
The optional sections about which to gather more specific information.

By default, a generic section of documentation is collected, whereas more specific information can be gathered for a specified problem type; for example, a section name of *trace* gathers all of the contents of the trace directory.

The default collections can be avoided by supplying a section name of *nodefault*.

IBM support generally supplies you with the sections to use. Example available sections are:

all Gathers all possible information, including all trace files, and diagnostics for many different types of problems. You must use this option only in certain circumstances and this option is not intended for general use.

default
IBM WebSphere MQ logs, FDC files, basic configuration, and status.

Note: Always gathered unless you use the section name **nodefault**.

nodefault
Prevents the default collections from occurring, but other explicitly requested sections are still collected.

trace Gathers all the trace file information plus the default information.

Note: Does not enable tracing.

defs Gathers the queue manager definitions and status information.

cluster
Gathers cluster configuration and queue information.

From IBM WebSphere MQ Version 7.5.0, Fix Pack 1 you can also specify the following sections:

dap Gathers transaction and persistence information.

kernel Gathers queue manager kernel information.

logger Gathers recovery logging information.

topic Gathers topic tree information.

From IBM WebSphere MQ Version 7.5.0, Fix Pack 2 you can specify the following section:

QMGR
Gathers all queue manager files: queues, logs, and configuration files.

For more information, see Section names and descriptions, in the IBM WebSphere MQ technote on using the IBM WebSphere MQ **runmqras** command to collect data.

-qmlist *QMA[,QMB]*
A list of queue manager names on which the **runmqras** command is to be run. This parameter does not apply to a client product (for example, HP Integrity NonStop Server) because there are no queue managers from which to request direct output.

By supplying a comma-separated list, you can restrict the iteration across queue managers to a specific list of queue managers. By default, iteration of commands is across all queue managers.

-timeout *secs*
The default timeout to give an individual command before the command stops waiting for completion.

By default, a timeout of 10 seconds is used. A value of zero means wait indefinitely.

-demo
Run in demonstration mode where no commands are processed, and no files gathered.

By running in demonstration mode, you can see exactly which commands would have been processed, and what files would have been gathered. The output zip file contains a `console.log` file that documents exactly what would have been processed and gathered, should the command be run normally.

-v Extends the amount of information that is logged in the `console.log` file, contained in the output zip file.

-ftp *ibm/custom*
Allows the collected archive to be sent through basic FTP to a remote destination.

At the end of processing, the resultant archive can be sent through basic FTP, either directly into IBM, or to a site of your choosing. If you select the *ibm* option, anonymous FTP is used to deliver the archive into the IBM ECuRep server. This process is identical to submitting the file manually using FTP.

Note if you select the *ibm* option, you must also provide the *pmrno* option, and all other FTP* options are ignored.

-ftpserver *server*
An FTP server name to connect to, when an FTP custom option is used.

-ftpusername *userid*
The user ID to log in to the FTP server with, when an FTP custom option is used.

-ftppassword *password*
The password to log in to the FTP server with, when an FTP custom option is used.

-ftpdirectory *path*
The directory on the FTP server to place the resulting zip file into, used when an FTP custom option is used.

-pmrno *12345,678,9AB*
A valid IBM PMR number (problem record number) against which to associate the documentation.

Use this option to ensure that the output is prefixed with your PMR Number, so that when the information is sent into IBM, the information is automatically associated with that problem record.

-help
Provide simple help.

-sub
Shows the keywords that will be substituted in the xml.

Examples

This command gathers the default documentation from the IBM WebSphere MQ installation, and all queue managers on a machine:

```
runmqras
```

This command gathers the default documentation from the IBM WebSphere MQ installation on a machine, and sends it directly into IBM to be associated with PMR number 11111,222,333 using the basic FTP capability:

```
runmqras -ftp ibm -pmrno 11111,222,333
```

This command gathers the default documentation from a machine, plus all trace files, the queue manager definitions, and status for all queue managers on the machine:

```
runmqras -section trace,defs
```

Return codes

A non zero return code indicates failure.

runmqsc:

Run WebSphere MQ commands on a queue manager.

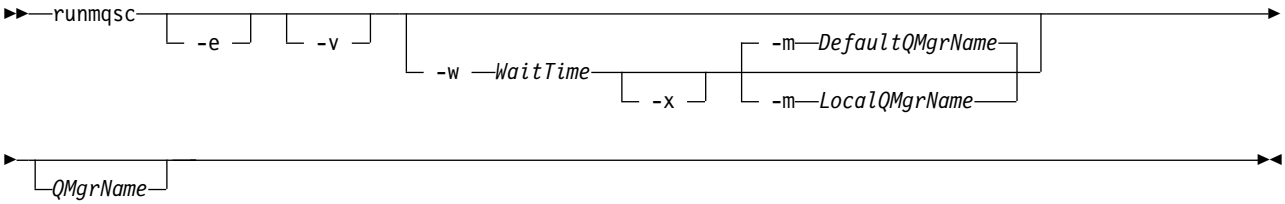
Purpose

Use the **runmqsc** command to issue MQSC commands to a queue manager. MQSC commands enable you to perform administration tasks, for example defining, altering, or deleting a local queue object. MQSC commands and their syntax are described in the MQSC reference.

You must use the **runmqsc** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

To end using the **runmqsc** command, use the **end** command. You can also use the **exit** or the **quit** command to stop**runmqsc**.

Syntax



Description

You can start the **runmqsc** command in three ways:

Verify command
Verify MQSC commands but do not run them. An output report is generated indicating the success or failure of each command. This mode is available on a local queue manager only.

Run command directly
Send MQSC commands directly to a local queue manager.

Run command indirectly
Run MQSC commands on a remote queue manager. These commands are put on the command queue on a remote queue manager and run in the order in which they were queued. Reports from the commands are returned to the local queue manager.

The **runmqsc** command takes its input from `stdin`. When the commands are processed, the results and a summary are put into a report that is sent to `stdout`.

By taking `stdin` from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file, you can run a sequence of frequently used commands contained in the file. You can also redirect the output report to a file.

Optional parameters

-e Prevents source text for the MQSC commands from being copied into a report. This parameter is useful when you enter commands interactively.

-m *LocalQMgrName*
The local queue manager that you want to use to submit commands to the remote queue manager. If you omit this parameter the local default queue manager is used to submit commands to the remote queue manager.

-v Verifies the specified commands without performing the actions. This mode is only available locally. The -w and -x flags are ignored if they are specified at the same time.

Important: The **-v** flag checks the syntax of the command only. Setting the flag does not check if any objects mentioned in the command actually exist.

For example, if the queue `Q1` does not exist in the queue manager, the following command is syntactically correct and does not generate any syntax errors: `runmqsc -v Qmgr display ql(Q1)`.

However, if you omit the **-v** flag, you receive error message AMQ8147.

-w *WaitTime*
Run the MQSC commands on another queue manager. You must have the required channel and transmission queues set up for this. See Preparing channels and transmission queues for remote administration for more information.

WaitTime
The time, in seconds, that **runmqsc** waits for replies. Any replies received after this are discarded, but the MQSC commands still run. Specify a time in the range 1 through 999 999 seconds.

Each command is sent as an Escape PCF to the command queue (SYSTEM.ADMIN.COMMAND.QUEUE) of the target queue manager.

The replies are received on queue SYSTEM.MQSC.REPLY.QUEUE and the outcome is added to the report. This can be defined as either a local queue or a model queue.

This flag is ignored if the -v flag is specified.

-x The target queue manager is running under z/OS. This flag applies only in indirect mode. The -w flag must also be specified. In indirect mode, the MQSC commands are written in a form suitable for the WebSphere MQ for z/OS command queue.

QMgrName
The name of the target queue manager on which to run the MQSC commands, by default, the default queue manager.

Return codes

Return code	Description
00	MQSC command file processed successfully
10	MQSC command file processed with errors; report contains reasons for failing commands
20	Error; MQSC command file not run

Examples

1. Enter this command at the command prompt:

   ```
   runmqsc
   ```

   Now you can enter MQSC commands directly at the command prompt. No queue manager name is specified, so the MQSC commands are processed on the default queue manager.
2. Use one of these commands, as appropriate in your environment, to specify that MQSC commands are to be verified only:

```
runmqsc -v BANK < "/u/users/commfile.in"

runmqsc -v BANK < "c:\users\commfile.in"
```

This command verifies the MQSC commands in file `commfile.in`. The queue manager name is `BANK`. The output is displayed in the current window.

3. These commands run the MQSC command file `mqscfile.in` against the default queue manager.

```
runmqsc < "/var/mqm/mqsc/mqscfile.in" > "/var/mqm/mqsc/mqscfile.out"

runmqsc < "c:\Program Files\IBM\WebSphere MQ\mqsc\mqscfile.in" >
 "c:\Program Files\IBM\WebSphere MQ\mqsc\mqscfile.out"
```

In this example, the output is directed to file `mqscfile.out`.

4. This command submits commands to the QMREMOTE queue manager, using QMLOCAL to submit the commands.

```
runmqsc -w 30 -m QMLOCAL QMREMOTE
```

runmqtmc:

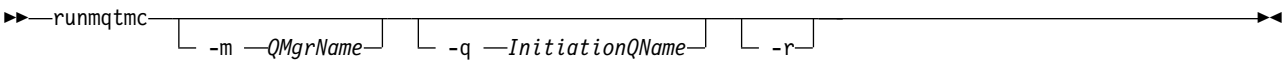
Start the trigger monitor on a client.

Purpose

Use the **runmqtmc** command to start a trigger monitor for a client. For further information about using trigger monitors, see Trigger monitors.

When a trigger monitor starts, it continuously monitors the specified initiation queue. The trigger monitor does not stop until the queue manager ends, see “endmqm” on page 189. While the client trigger monitor is running it keeps the dead letter queue open.

Syntax



Optional parameters

-m *QMgrName*
The name of the queue manager on which the client trigger monitor operates, by default the default queue manager.

-q *InitiationQName*
The name of the initiation queue to be processed, by default SYSTEM.DEFAULT.INITIATION.QUEUE.

-r
Specifies that the client trigger monitor automatically reconnects.

Return codes

Return code	Description
0	Not used. The client trigger monitor is designed to run continuously and therefore not to end. The value is reserved.
10	Client trigger monitor interrupted by an error.
20	Error; client trigger monitor not run.

Examples

For examples of using this command, see The Triggering sample programs.

runmqtrm:

Start the trigger monitor on a server.

Purpose

Use the **runmqtrm** command to start a trigger monitor. For further information about using trigger monitors, see Trigger monitors.

When a trigger monitor starts, it continuously monitors the specified initiation queue. The trigger monitor does not stop until the queue manager ends, see “endmqm” on page 189. While the trigger monitor is running it keeps the dead letter queue open.

Syntax

Optional parameters

-m *QMgrName*
The name of the queue manager on which the trigger monitor operates, by default the default queue manager.

-q *InitiationQName*
Specifies the name of the initiation queue to be processed, by default SYSTEM.DEFAULT.INITIATION.QUEUE.

Return codes

Return code	Description
0	Not used. The trigger monitor is designed to run continuously and therefore not to end. Hence a value of 0 would not be seen. The value is reserved.
10	Trigger monitor interrupted by an error.
20	Error; trigger monitor not run.

runswchl:

runswchl (switch cluster channel) on UNIX, Linux, and Windows.

Purpose

The command switches or queries the cluster transmission queues associated with cluster-sender channels.

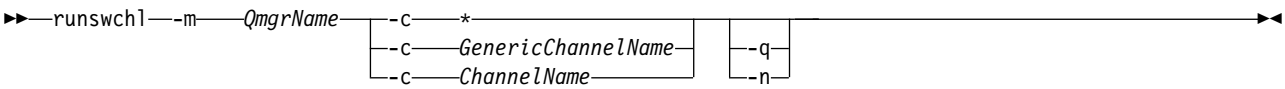
Usage notes

You must log on as an Administrator to run this command.

The command switches all the stopped or inactive cluster-sender channels that match the **-c** parameter, require switching, and can be switched. The command reports back on the channels that are switched, the channels that do not require switching, and the channels it cannot switch because they are not stopped or inactive.

If you set the **-q** parameter, the command does not perform the switch, but it provides the list of channels that would be switched.

Syntax



Required parameters

-m *QmgrName*
The queue manager to run the command against. The queue manager must be started.

-c *
All the cluster-sender channels

-c *GenericChannelName*
All matching cluster-sender channels

-c *ChannelName*
Single cluster-sender channel.

Optional parameters

-q Display the state of one or more channels. If you omit this parameter, the commands switches any stopped or inactive channels that require switching.

-n When switching transmission queues, do not transfer messages from the old queue to the new transmission queue.

Note: Take care with the **-n** option: messages on the old transmission queue are not transferred unless you associate the transmission queue with another cluster-sender channel.

Return codes

0 The command completed successfully

10 The command completed with warnings.

20 The command completed with errors.

Examples

To display the configuration state of cluster-sender channel `TO.QM2`:

```
RUNSWCHL -m QM1 -c TO.QM2 -q
```

To switch the transmission queue for cluster-sender channel `TO.QM3` without moving the messages on it:

```
RUNSWCHL -m QM1 -c TO.QM3 -n
```

To switch the transmission queue for cluster-sender channel TO.QM3 and move the messages on it:

```
RUNSWCHL -m QM1 -c TO.QM3
```

To display the configuration state of all cluster-sender channels on QM1:

```
RUNSWCHL -m QM1 -c * -q
```

To display the configuration state of all cluster-sender channels with a generic name of TO.*:

```
RUNSWCHL -m QM1 -c TO.* -q
```

Related information:

Clustering: Switching cluster transmission queues

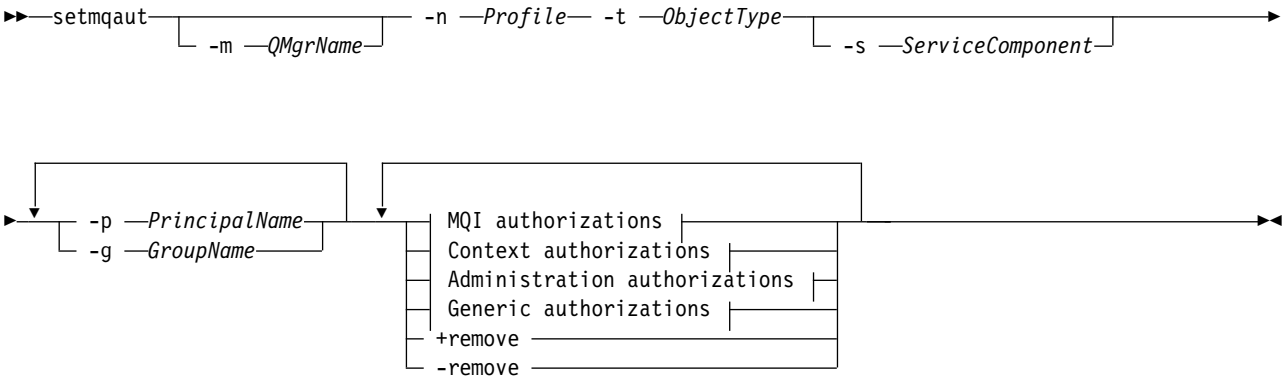
setmqaut:

Change the authorizations to a profile, object, or class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

For more information about authorization service components, see Installable services, Service components, and Authorization service interface.

For more information about how authorizations work, see How authorizations work.

Syntax



MQI authorizations:

Context authorizations:

Administration authorizations:

Generic authorizations:

Description

Use **setmqaut** both to *grant* an authorization, that is, give a principal or user group permission to perform an operation, and to *revoke* an authorization, that is, remove the permission to perform an operation. You can specify a number of parameters:

- Queue manager name
- Principals and user groups
- Object type
- Profile name
- Service component

The authorizations that can be given are categorized as follows:

- Authorizations for issuing MQI calls
- Authorizations for MQI context
- Authorizations for issuing commands for administration tasks
- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by a plus sign (+) or a minus sign (-). For example, if you include +put in the authorization list, you grant authority to issue MQPUT calls against a queue. Alternatively, if you include -put in the authorization list, you revoke the authority to issue MQPUT calls.

You can specify any number of principals, user groups, and authorizations in a single command, but you must specify at least one principal or user group.

If a principal is a member of more than one user group, the principal effectively has the combined authorities of all those user groups. On Windows systems, the principal also has all the authorities that have been granted to it explicitly using the **setmqaut** command.

On UNIX systems, all authorities are held by user groups internally, not by principals. Granting authorities to groups has the following implications:

- If you use the **setmqaut** command to grant an authority to a principal, the authority is granted to the primary user group of the principal. This means that the authority is effectively granted to all members of that user group.
- If you use the **setmqaut** command to revoke an authority from a principal, the authority is revoked from the primary user group of the principal. This means that the authority is effectively revoked from all members of that user group.

To alter authorizations for a cluster sender channel that has been automatically generated by a repository, see Channel definition commands.

Required parameters

-t *ObjectType*
The type of object for which to change authorizations.

Possible values are as follows:

authinfo	An authentication information object
channel or **chl**	A channel
clntconn or **clcn**	A client connection channel
comminfo	A communication information object
listener or **lstr**	A listener
namelist or **nl**	A namelist
process or **prcs**	A process
queue or **q**	A queue
qmgr	A queue manager
rqmname or **rqmn**	A remote queue manager name
service or **srvc**	A service
topic or **top**	A topic

-n *Profile*
The name of the profile for which to change authorizations. The authorizations apply to all WebSphere MQ objects with names that match the profile name specified. The profile name can be generic, using wildcard characters to specify a range of names as explained in Using OAM generic profiles on UNIX or Linux systems and Windows.

This parameter is required, unless you are changing the authorizations of a queue manager, in which case you must *not* include it. To change the authorizations of a queue manager use the queue manager name, for example

```
setmqaut -m QMGR -t qmgr -p user1 +connect
```

where *QMGR* is the name of the queue manager and *user1* is the user requesting the change.

Each class of object has authority records for each group or principal. These records have the profile name `@CLASS` and track the `crt` (create) authority common to all objects of that class. If the `crt` authority for any object of that class is changed then this record is updated. For example:

```
profile:     @class
object type: queue
entity:      test
entity type: principal
authority:   crt
```

This shows that members of the group `test` have `crt` authority to the class `queue`.

Optional parameters

-m *QMgrName*
The name of the queue manager of the object for which to change authorizations. The name can contain up to 48 characters.

This parameter is optional if you are changing the authorizations of your default queue manager.

-p *PrincipalName*
The name of the principal for which to change authorizations.

For WebSphere MQ for Windows only, the name of the principal can optionally include a domain name, specified in the following format:

```
userid@domain
```

For more information about including domain names on the name of a principal, see Principals and groups .

You must have at least one principal or group.

-g *GroupName*
The name of the user group for which to change authorizations. You can specify more than one group name, but each name must be prefixed by the -g flag.

For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

The WebSphere MQ Object Authority Manager validates the users and groups at the domain level, only if you set the **GroupModel** attribute to *GlobalGroups* in the Security stanza of the queue manager.

-s *ServiceComponent*
The name of the authorization service to which the authorizations apply (if your system supports installable authorization services). This parameter is optional; if you omit it, the authorization update is made to the first installable component for the service.

+remove or -remove
Remove all the authorities from WebSphere MQ objects that match the specified profile.

Authorizations
The authorizations to be granted or revoked. Each item in the list is prefixed by a plus sign (+) or a minus sign (-). The plus sign indicates that authority is to be granted. The minus sign indicates that authority is to be revoked.

For example, to grant authority to issue MQPUT calls, specify +put in the list. To revoke the authority to issue MQPUT calls, specify -put.

Table 48 shows the authorities that can be given to the different object types.

Table 48. Specifying authorities for different object types.

Cross-tabulation of object types versus authority. Each cell contains whether the authority can be given to the object type.

Authority	Queue	Process	Queue manager	Remote queue manager name	Namelist	Topic	Auth info	Clntconn	Channel	Listener	Service	Comminfo
all [1]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
alladm [2]	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
allmqi [3]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No
none	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes
altusr	No	No	Yes	No	No	No	No	No	No	No	No	No
browse	Yes	No	No	No	No	No	No	No	No	No	No	No
chg	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
clr	Yes	No	No	No	No	Yes	No	No	No	No	No	No

Table 48. Specifying authorities for different object types (continued).

Cross-tabulation of object types versus authority. Each cell contains whether the authority can be given to the object type.

Authority	Queue	Process	Queue manager	Remote queue manager name	Namelist	Topic	Auth info	Clntconn	Channel	Listener	Service	Comminfo
connect	No	No	Yes	No	No	No	No	No	No	No	No	No
crt	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
ctrl	No	No	No	No	No	Yes	No	No	Yes	Yes	Yes	No
ctrlx	No	No	No	No	No	No	No	No	Yes	No	No	No
dlt	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
dsp	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
get	Yes	No	No	No	No	No	No	No	No	No	No	No
pub	No	No	No	No	No	Yes	No	No	No	No	No	No
put	Yes	No	No	Yes	No	No	No	No	No	No	No	No
inq	Yes	Yes	Yes	No	Yes	No	Yes	No	No	No	No	No
passall	Yes	No	No	No	No	No	No	No	No	No	No	No
passid	Yes	No	No	No	No	No	No	No	No	No	No	No
resume	No	No	No	No	No	Yes	No	No	No	No	No	No
set	Yes	Yes	Yes	No	No	No	No	No	No	No	No	No
setall	Yes	No	Yes	No	No	No	No	No	No	No	No	No
setid	Yes	No	Yes	No	No	Yes	No	No	No	No	No	No
sub	No	No	No	No	No	Yes	No	No	No	No	No	No
system	No	No	Yes	No	No	No	No	No	No	No	No	No

Note:

1. `all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.
2. `alladm` authority is equivalent to the union of the individual authorities `chg`, `clr`, `dlt`, `dsp`, `ctrl`, and `ctrlx` appropriate to the object type. `crt` authority is not included in the subset `alladm`.
3. `allmqi` authority is equivalent to the union of the individual authorities `altusr`, `browse`, `connect`, `get`, `inq`, `pub`, `put`, `resume`, `set`, and `sub` appropriate to the object type.

Description of specific authorities

You should not grant a user an authority (for example, `set` authority on a queue manager, or `system` authority) that allows the user to access WebSphere MQ privileged options, unless the required authority is specifically documented, and required to run any WebSphere MQ command, or WebSphere MQ API call.

For example, a user requires system authority to run the **setmqaut** command.

chg

A user needs `chg` authority to make any authorization changes on the queue manager. The authorization changes include:

- Changing the authorizations to a profile, object, or class of objects
- Creating and modifying channel authentication records, and so on

A user also needs chg authority to change or set the attributes of an WebSphere MQ object, using PCF or MQSC commands.

crt

If you grant an entity +crt authority to the queue manager, then that entity also gains +crt authority for each object class.

However, when you remove +crt authority against the queue manager object that only removes the authority on the queue manager object class; crt authority for other objects classes are not removed.

Note that crt authority on the queue manager object has no functional use, and is available for backwards-compatibility purposes only.

dlt

Note that the dlt authority against the queue manager object has no functional use, and is available for backwards-compatibility purposes only.

set

A user needs set authority against the queue to change or set the attributes of a queue using the MQSET API call.

set authority on the queue manager is not required for any administrative purpose, or for any application connecting to the queue manager.

However, a user needs set authority against the queue manager to set privileged connection options.

Note that set authority on the process object has no functional use, and is available for backwards-compatibility purposes only.

Important: Privileged connection options are internal to the queue manager and are not available in WebSphere MQ API calls used by WebSphere MQ applications.

system

The **setmqaut** command makes a privileged WebSphere MQ connection to the queue manager.

Any user who runs WebSphere MQ commands that makes a privileged WebSphere MQ connection needs system authority on the queue manager.

Return codes

Return code	Explanation
0	Successful operation
26	Queue manager running as a standby instance.
36	Invalid arguments supplied
40	Queue manager not available
49	Queue manager stopping
58	Inconsistent use of installations detected
69	Storage not available
71	Unexpected error
72	Queue manager name error
133	Unknown object name
145	Unexpected object name
146	Object name missing
147	Object type missing
148	Invalid object type
149	Entity name missing
150	Authorization specification missing

Return code	Explanation
151	Invalid authorization specification

Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue orange.queue on queue manager saturn.queue.manager.

   ```
   setmqaut -m saturn.queue.manager -n orange.queue -t queue
            -g tango +inq +alladm
   ```

   The authorizations are given to a user group called tango, and the associated authorization list specifies that the user group can:
   - Issue MQINQ calls
   - Perform all administration operations on that object
2. In this example, the authorization list specifies that a user group called foxy:
   - Cannot issue any MQI calls to the specified queue
   - Can perform all administration operations on the specified queue

   ```
   setmqaut -m saturn.queue.manager -n orange.queue -t queue
            -g foxy -allmqi +alladm
   ```
3. This example gives user1 full access to all queues with names beginning a.b. on queue manager qmgr1. The profile applies to any object with a name that matches the profile.

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 +all
   ```
4. This example deletes the specified profile.

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 -remove
   ```
5. This example creates a profile with no authority.

   ```
   setmqaut -m qmgr1 -n a.b.* -t q -p user1 +none
   ```

Related reference:

"SET AUTHREC" on page 767
Use the MQSC command SET AUTHREC to set authority records associated with a profile name.

Authorizations for MQI calls:

altusr	Use another user's authority for the queue manager. Also required for channel operations where the asserting userid is different from the one associated with the connection handle. (For example, an assigned dedicated profile on the receiver MCA end, or when processing a RESET CHL SEQNUM() request from remote systems.) If you are using WebSphere MQ prior to version 7.0.1.4, you must set +altusr for the group containing the user ID specified in MCAUSER on a receiver channel. This action prevents error message AMQ2035 appearing if you reset the sequence number of the corresponding sender channel.
browse	Retrieve a message from a queue using an MQGET call with the BROWSE option.
connect	Connect the application to the specified queue manager using an MQCONN call.
get	Retrieve a message from a queue using an MQGET call.
inq	Make an inquiry on a specific queue using an MQINQ call.
pub	Publish a message on a topic using the MQPUT call.
put	Put a message on a specific queue using an MQPUT call.
resume	Resume a subscription using the MQSUB call.
set	Set attributes on a queue from the MQI using an MQSET call.
sub	Create, alter or resume a subscription to a topic using the MQSUB call.

Note: If you open a queue for multiple options, you must be authorized for each option.

Authorizations for context:

passall	Pass all context on the specified queue. All the context fields are copied from the original request.
passid	Pass identity context on the specified queue. The identity context is the same as that of the request.
setall	Set all context on the specified queue. This is used by special system utilities.
setid	Set identity context on the specified queue. This is used by special system utilities.
	In order to modify any of the message context options, you must have the appropriate authorizations to issue the call. For example, in order to use MQOO_SET_IDENTITY_CONTEXT or MQPMO_SET_IDENTITY_CONTEXT, you must have `+setid` permission.

Note: To use setid or setall authority authorizations must be granted on both the appropriate queue object and also on the queue manager object.

Authorizations for commands:

chg	Change the attributes of the specified object.
clr	Clear the specified queue or a topic.
crt	Create objects of the specified type.
dlt	Delete the specified object. Note, that the dlt authority has no effect on a queue manager object.
dsp	Display the attributes of the specified object.
ctrl	For listeners and services, start and stop the specified channel, listener, or service. For channels, start, stop, and ping the specified channel. For topics, define, alter, or delete subscriptions.
ctrlx	Reset or resolve the specified channel.

Authorizations for generic operations:

all	Use all operations applicable to the object. `all` authority is equivalent to the union of the authorities `alladm`, `allmqi`, and `system` appropriate to the object type.
alladm	Use all administration operations applicable to the object.
allmqi	Use all MQI calls applicable to the object.
none	No authority. Use this authorization to create profiles without authority. When an authority is given to an object or group that was previously showing "none", then the authorization changes to the authority just applied. However, when the "none" authorization is added to an object or group with an existing alternative authority, the authority does not change.
system	Use queue manager for internal system operations.

setmqcrl:

Administer CRL (certificate revocation list) LDAP definitions in an Active Directory (Windows only).

Purpose

Note: The **setmqcrl** command applies to WebSphere MQ for Windows only.

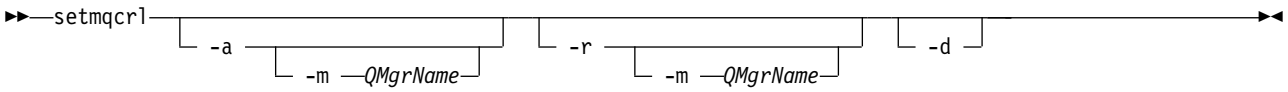
Use the **setmqcrl** command to configure and administer support for publishing CRL (certificate revocation list) LDAP definitions in an Active Directory.

A domain administrator must use this command, or **setmqscp**, initially to prepare the Active Directory for WebSphere MQ usage and to grant WebSphere MQ users and administrators the relevant authorities to access and update the WebSphere MQ Active Directory objects. You can also use the **setmqcrl**

command to display all the currently configured CRL server definitions available on the Active Directory, that is, those definitions referred to by the queue manager's CRL namelist.

The only types of CRL servers supported are LDAP servers.

Syntax



Optional parameters

You must specify one of `-a` (add), `-r` (remove) or `-d` (display).

-a Adds the WebSphere MQ MQI client connections Active Directory container, if it does not already exist. You must be a user with the appropriate privileges to create subcontainers in the *System* container of your domain. The WebSphere MQ folder is called `CN=IBM-MQClientConnections`. Do not delete this folder in any other way than by using the **setmqscp** command.

-d Displays the WebSphere MQ CRL server definitions.

-r Removes the WebSphere MQ CRL server definitions.

-m [* | qmgr]
Modifies the specified parameter (`-a` or `-r`) so that only the specified queue manager is affected. You must include this option with the `-a` parameter.

*** | qmgr**
* specifies that all queue managers are affected. This enables you to migrate a specific WebSphere MQ CRL server definitions file from one queue manager alone.

Examples

The following command creates the IBM-MQClientConnections folder and allocates the required permissions to WebSphere MQ administrators for the folder, and to child objects created subsequently. (In this, it is functionally equivalent to `setmqscp -a`.)

```
setmqcrl -a
```

The following command migrates existing CRL server definitions from a local queue manager, Paint.queue.manager, to the Active Directory, **deleting any other CRL definitions from the Active Directory first**:

```
setmqcrl -a -m Paint.queue.manager
```

setmqenv:

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

Purpose

You can use the **setmqenv** script to automatically set up the environment for use with an installation of IBM WebSphere MQ. Alternatively, you can use the **crtmqenv** command to create a list of environment variables and values to manually set each environment variable for your system; see “crtmqenv” on page 142 for more information.

Note: Any changes you make to the environment are not persistent. If you log out, and log in again, your changes are lost.

You can specify which installation the environment is set up for by specifying a queue manager name, an installation name, or an installation path. You can also set up the environment for the installation that issues the **setmqenv** command by issuing the command with the **-s** parameter.

The **setmqenv** command sets the following environment variables, appropriate to your system:

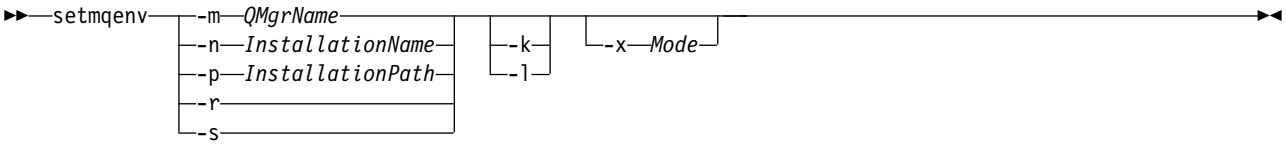
- CLASSPATH
- INCLUDE
- LIB
- MANPATH
- MQ_DATA_PATH
- MQ_ENV_MODE
- MQ_FILE_PATH
- MQ_JAVA_INSTALL_PATH
- MQ_JAVA_DATA_PATH
- MQ_JAVA_LIB_PATH
- MQ_JAVA_JVM_FLAG
- MQ_JRE_PATH
- PATH

On UNIX and Linux systems, if the **-l** or **-k** flag is specified, the *LIBPATH* environment variable is set on AIX, and the *LD_LIBRARY_PATH* environment variable is set on HP-UX, Linux, and Solaris.

Usage notes

- If you have installed IBM WebSphere MQ V 7.0.1, do not use the **setmqenv** command. Some of the components of IBM WebSphere MQ V 7.0.1, such as Explorer, reference the environment variables for their library paths and therefore will not work if the **setmqenv** command has been used to alter the environment variables to point to a IBM WebSphere MQ V 7.0.1 installation path.
- The **setmqenv** command removes all directories for all IBM WebSphere MQ installations from the environment variables before adding new references to the installation for which you are setting up the environment for. Therefore, if you want to set any additional environment variables that reference IBM WebSphere MQ, set the variables after issuing the **setmqenv** command. For example, if you want to add *MQ_INSTALLATION_PATH*`/java/lib` to *LD_LIBRARY_PATH*, you must do so after running the **setmqenv** command.
- In some shells, command-line parameters cannot be used with **setmqenv** and any **setmqenv** command issued is assumed to be a `setmqenv -s` command. The command produces an informational message that the command has been run as if a `setmqenv -s` command had been issued. Therefore, in these shells you must ensure that you issue the command from the installation for which you want to set the environment for. In these shells, you must set the *LD_LIBRARY_PATH* variable manually. Use the **crtmqenv** command with the **-l** or **-k** parameter to list the *LD_LIBRARY_PATH* variable and value. Then use this value to set the *LD_LIBRARY_PATH*.

Syntax



Optional Parameters

-m ***QMgrName***
Set the environment for the installation associated with the queue manager *QMgrName*.

-n ***InstallationName***
Set the environment for the installation named *InstallationName*.

-p ***InstallationPath***
Set the environment for the installation in the path *InstallationPath*.

-r Remove all installations from the environment.

-s Set the environment for the installation that issued the **setmqenv** command.

-k UNIX and Linux only.

Include the *LD_LIBRARY_PATH* or *LIBPATH* environment variable in the environment, adding the path to the IBM WebSphere MQ libraries at the start of the current *LD_LIBRARY_PATH* or *LIBPATH* variable.

-l UNIX and Linux only.

Include the *LD_LIBRARY_PATH* or *LIBPATH* environment variable in the environment, adding the path to the IBM WebSphere MQ libraries at the end of the current *LD_LIBRARY_PATH* or *LIBPATH* variable.

-x ***Mode***
Mode can take the value 32 or 64.

Create a 32-bit or 64-bit environment. If this parameter is not specified, the environment matches that of the queue manager or installation specified in the command.

Any attempt to display a 64-bit environment with a 32-bit installation fails.

Return codes

Return code	Description
0	Command completed normally.
10	Command completed with unexpected results.
20	An error occurred during processing.

Examples

The following examples assume that a copy of IBM WebSphere MQ is installed in the `/opt/mqm` directory on a UNIX or Linux system.

Note: The period character (`.`) character used at the beginning of each command makes the **setmqenv** script run in the current shell. Therefore, the environment changes made by the **setmqenv** script are applied to the current shell. Without the period character (`.`), the environment variables are changed in another shell, and the changes are not applied to the shell from which the command is issued.

- The following command sets up the environment for an installation installed in the `/opt/mqm` directory:
  ```
  . /opt/mqm/bin/setmqenv -s
  ```
- The following command sets up the environment for an installation installed in the `/opt/mqm2` directory, and includes the path to the installation at the end of the current value of the *LD_LIBRARY_PATH* variable:
  ```
  . /opt/mqm/bin/setmqenv -p /opt/mqm2 -l
  ```
- The following command sets up the environment for queue manager `QM1` in a 32-bit environment:
  ```
  . /opt/mqm/bin/setmqenv -m QM1 -x 32
  ```

The following example assumes that a copy of IBM WebSphere MQ is installed in `C:\Program Files\IBM\WebSphere MQ` on a Windows system.

This command sets up the environment for an installation called `Installation1`:

```
"C:\Program Files\IBM\WebSphere MQ\bin\setmqenv.cmd" -n Installation1
```

Related reference:

"crtmqenv" on page 142
Create a list of environment variables for an installation of IBM WebSphere MQ, on UNIX, Linux, and Windows.

Related information:

Choosing a primary installation

Multiple installations

setmqinst:

Set IBM WebSphere MQ installations, on UNIX, Linux, and Windows.

Purpose

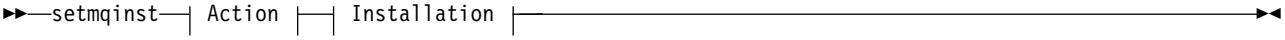
You can use the **setmqinst** command to change the installation description of an installation, or to set or unset an installation as the primary installation. To change the primary installation, you must unset the current primary installation before you can set a new primary installation. This command updates information contained in the `mqinst.ini` file.

After unsetting the primary installation, the **setmqinst** command will not be available unless you specify the full path or have an appropriate installation directory on your `PATH` (or equivalent). The default path in a system standard location will have been deleted.

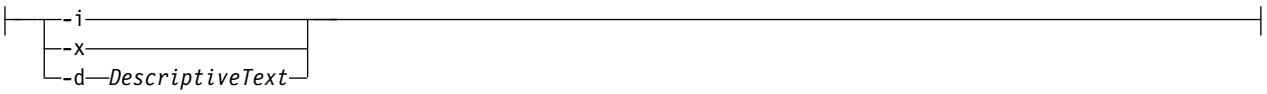
File `mqinst.ini` contains information about all IBM WebSphere MQ installations on a system. For more information about mqinst.ini, see Installation configuration file, mqinst.ini .

On UNIX or Linux systems, you must run this command as root. On Windows systems, you must run this command as a member of the Administrators group. The command does not have to be run from the installation you are modifying.

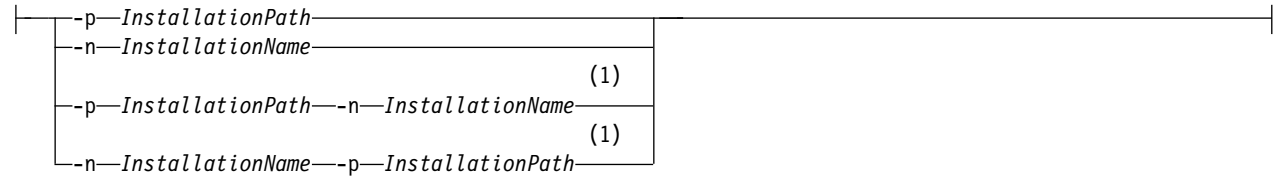
Syntax



Action:



Installation:



Notes:

1 When specified together, the installation name and installation path must refer to the same installation.

Parameters

-d *DescriptiveText*
Text that describes the installation.

The text can be up to 64 single-byte characters, or 32 double-byte characters. The default value is all blanks. You must use double quotation marks around the text if it contains spaces.

-i Set this installation as the primary installation.

-x Unset this installation as the primary installation.

-n *InstallationName*
The name of the installation to modify.

-p *InstallationPath*
The path of the installation to modify. You must use double quotation marks around the path if it contains spaces

Return codes

Return code	Description
0	Entry set without error
36	Invalid arguments supplied
37	Descriptive text was in error
44	Entry does not exist
59	Invalid installation specified
71	Unexpected error
89	ini file error
96	Could not lock ini file
98	Insufficient authority to access ini file
131	Resource problem

Examples

1. This command sets the installation with the name of myInstallation as the primary installation:
   ```
   setmqinst -i -n myInstallation
   ```
2. This command sets the installation with an installation path of `/opt/myInstallation` as the primary installation:
   ```
   setmqinst -i -p /opt/myInstallation
   ```
3. This command unsets the installation named myInstallation as the primary installation:
   ```
   setmqinst -x -n myInstallation
   ```
4. This command unsets the installation with an installation path of `/opt/myInstallation` as the primary installation:

```
setmqinst -x -p /opt/myInstallation
```

5. This command sets the descriptive text for the installation named myInstallation:

```
setmqinst -d "My installation" -n myInstallation
```

The descriptive text is enclosed in quotation marks as it contains spaces.

Related information:

Choosing a primary installation

Changing the primary installation

setmqm:

Set the associated installation of a queue manager.

Purpose

Use the **setmqm** command to set the associated IBM WebSphere MQ installation of a queue manager. The queue manager can then be administered using only the commands of the associated installation. For example, when a queue manager is started with **strmqm**, it must be the **strmqm** command of the installation that was specified by the **setmqm** command.

For more information about using this command, including information about when to use it, see Associating a queue manager with an installation.

This command is only applicable to UNIX, Linux and Windows.

Usage notes

- You must use the **setmqm** command from the installation you want to associate the queue manager with.
- The installation name specified by the **setmqm** command must match the installation from which the **setmqm** command is issued.
- You must stop the queue manager before executing the **setmqm** command. The command fails if the queue manager is still running.
- Once you have set the associated installation of a queue manager using the **setmqm** command, migration of the queue manager's data occurs when you start the queue manager using the **strmqm** command.
- Once you have started the queue manager on an installation, you cannot then use **setmqm** to set the associated installation to an earlier version of IBM WebSphere MQ, as it is not possible to migrate back to earlier versions of IBM WebSphere MQ.
- You can find out which installation is associated with a queue manager by using the **dspmq** command. See “dspmq” on page 163 for more information.

Syntax

Required Parameters

-m *QMgrName*
The name of the queue manager to set the associated installation for.

-n *InstallationName*
The name of the installation that the queue manager is to be associated with. The installation name is not case-sensitive.

Return codes

Return code	Description
0	Queue manager set to an installation without error
5	Queue manager running
36	Invalid arguments supplied
59	Invalid installation specified
60	Command not executed from the installation named by the -n parameter
61	Invalid installation name for this queue manager
69	Resource problem
71	Unexpected error
72	Queue manager name error
119	User not authorized

Examples

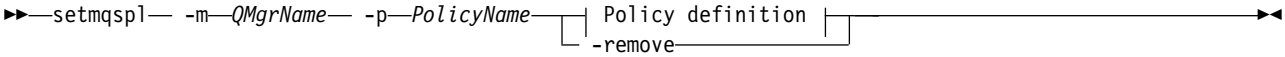
1. This command associates a queue manager QMGR1, with an installation with the installation name of myInstallation.

   ```
   MQ_INSTALLATION_PATH/bin/setmqm -m QMGR1 -n myInstallation
   ```

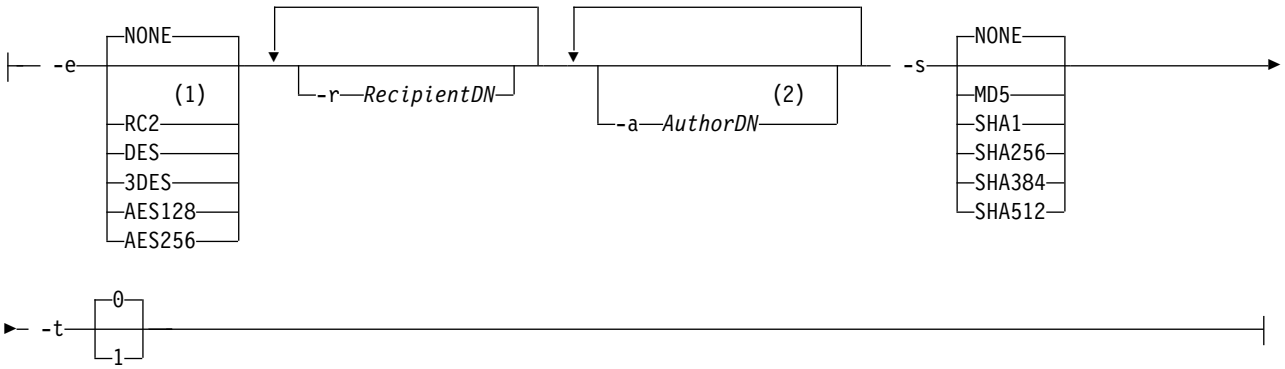
setmqspl:

Use the **setmqspl** command to define a new security policy, alter an already existing one, or remove an existing policy.

Syntax



Policy definition:



Notes:

1 If an encryption algorithm is selected, a recipient DN must also be provided.

2 If an author DN is provided, a signing algorithm must also be selected.

Table 49. `setmqspl` command flags.

Command flag	Explanation
`-m`	Queue manager name. This flag is mandatory for all actions on security policies.
`-p`	Policy name. Set the policy name to the name of the queue you wish the policy to apply to.
`-s`	Digital signature algorithm. WebSphere MQ Advanced Message Security supports the following values: `MD5`, `SHA1`, `SHA256`, `SHA384`, and `SHA512`. All must be in uppercase. The default value is `NONE` **Important:** • For the SHA384 and SHA512 cryptographic hash functions, keys used for signing must be longer than 768 bits. • Encryption algorithms' name must be in uppercase
`-e`	Digital encryption algorithm. WebSphere MQ Advanced Message Security supports the following encryption algorithms: `RC2`, `DES`, `3DES`, `AES128`, `AES256`. The default value is `NONE`. **Important:** Encryption algorithms' name must be in uppercase
`-r`	Message recipient's distinguished name (DN) (certificate of a DN provided is used to encrypt a given message). Recipients can be specified only if encryption algorithm is different from `NONE`. Multiple recipients can be included for a message. Each DN must be provided with a separate `-r` flag. **Important:** • DN attribute names must be in uppercase. • Commas must be used as a name separators. • To avoid command interpreter errors, place quotation marks around the DNs. For example: `-r "CN=alice, O=ibm, C=US"`
`-a`	Signature DN that is validated during message retrieval. Only messages signed by a user with a DN provided are accepted during the retrieval. Signature DNs can be specified only if the signature algorithm is different from `NONE`. Multiple authors can be included. Each author needs to have a separate `-a` flag. **Important:** DN attribute name must be in uppercase.

Table 49. `setmqspl` command flags. (continued)

Command flag	Explanation
`-t`	Toleration flag that indicates whether a policy that is associated with a queue can be ignored when an attempt to retrieve a message from the queue involves a message with no security policy set. Valid values include: • **0 (default)** Toleration flag off. • **1** Toleration flag on. Toleration is optional and facilitates staged roll-out, where policies were applied to queues but those queues may already contain messages that have no policy, or still receive messages from remote systems that do not have the security policy set.
`-remove`	Delete policy. If specified, only the `-p` flag remains valid.

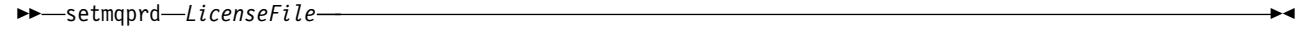
setmqprd:

Enroll a IBM WebSphere MQ production license.

A license is normally enrolled as part of the installation process.

Note: You must have the appropriate privileges to run this command on your system. UNIX requires root access, and Windows with UAC (User Account Control) requires Administrator access to run this command.

Syntax



Required parameters

LicenseFile
Specifies the fully-qualified name of the production license certificate file.

The full license file is **amqpcert.lic**. On UNIX and Linux, it is in the */MediaRoot*/licenses directory on the installation media. On Windows it is in the *MediaRoot*\licenses directory on the installation media. It is installed into the `bin` directory on the IBM WebSphere MQ installation path.

Trial license conversion

A trial license installation is identical to a production license installation, except for the "count-down" message that is displayed when you start a queue manager on an installation with a trial license. Parts of IBM WebSphere MQ that are not installed on the server, such as the IBM WebSphere MQ MQI client, continue to work after the expiry of the trial license. You do not need to run **setmqprd** to enroll them with a production license.

When a trial license expires, you can still uninstall IBM WebSphere MQ. You can also reinstall IBM WebSphere MQ with a full production license.

Run **setmqprd** to enroll a production license after installing and using a installation with a trial license.

Related information:

Converting a trial license on UNIX, Linux, and Windows

setmqscp:

Publish client connection channel definitions in an Active Directory (Windows only).

Purpose

Note: The **setmqscp** command applies to WebSphere MQ for Windows only.

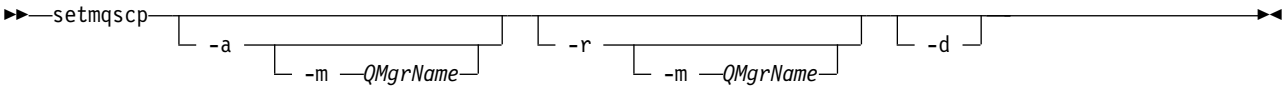
Use the **setmqscp** command to configure and administer support for publishing client connection channel definitions in an Active Directory.

Initially, this command is used by a domain administrator to:

- Prepare the Active Directory for WebSphere MQ use
- Grant WebSphere MQ users and administrators the relevant authorities to access and update the WebSphere MQ Active Directory objects

You can also use the **setmqscp** command to display all the currently configured client connection channel definitions available on the Active Directory.

Syntax



Optional parameters

You must specify one of -a (add), -r (remove) or -d (display).

-a Adds the WebSphere MQ MQI client connections Active Directory container, if it does not already exist. You must be a user with the appropriate privileges to create subcontainers in the *System* container of your domain. The WebSphere MQ folder is called `CN=IBM-MQClientConnections`. Do not delete this folder in any other way than by using the `setmqscp -r` command.

-d Displays the service connection points.

-r Removes the service connection points. If you omit -m, and no client connection definitions exist in the IBM-MQClientConnections folder, the folder itself is removed from the Active Directory.

-m [* | qmgr]
Modifies the specified parameter (-a or -r) so that only the specified queue manager is affected.

*** | qmgr**
* specifies that all queue managers are affected. This enables you to migrate a specific client connection table file from one queue manager alone, if required.

Examples

The following command creates the IBM-MQClientConnections folder and allocates the required permissions to WebSphere MQ administrators for the folder, and to child objects created subsequently:

```
setmqscp -a
```

The following command migrates existing client connection definitions from a local queue manager, Paint.queue.manager, to the Active Directory:

```
setmqscp -a -m Paint.queue.manager
```

The following command migrates all client connection definitions on the local server to the Active Directory:

```
setmqscp -a -m *
```

strmqcfg:

Start IBM WebSphere MQ Explorer (Windows, Linux x86, and Linux x86-64 platforms only).

Purpose

For IBM WebSphere MQ for Windows only, note that if you use **runas** to execute this command, you must define the Environment Variable *APPDATA* to set a path to a directory that the user you are running as has access. For example:

```
set APPDATA=C:\Users\user_name\AppData\Roaming
```

You can use the following command to identify the path that *APPDATA* is set to:

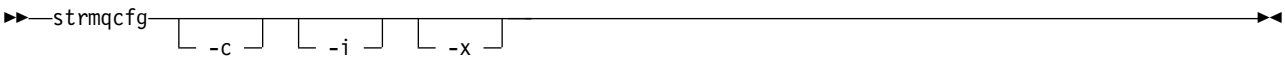
```
set APPDATA
```

On Linux, to start IBM WebSphere MQ Explorer successfully, you must be able to write a file to your home directory, and the home directory must exist.

Note: The preferred way to start IBM WebSphere MQ Explorer on Windows and Linux systems is by using the system menu, or the `MQExplorer` executable file.

Syntax

The syntax of this command follows:



Optional parameters

-c `-clean` is passed to Eclipse. This parameter causes Eclipse to delete any cached data used by the Eclipse runtime.

-i `-clean -initialize` is passed to Eclipse. This parameter causes Eclipse to delete any cached data as well as discard configuration information used by the Eclipse runtime. IBM WebSphere MQ Explorer starts briefly and then ends without displaying the user interface.

-x Output debug messages to the console.

strmqcsv:

Start the command server for a queue manager.

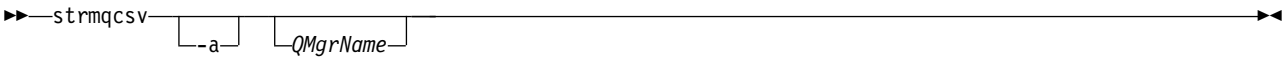
Purpose

Use the **strmqcsv** command to start the command server for the specified queue manager. This enables WebSphere MQ to process commands sent to the command queue.

You must use the **strmqcsv** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

If the queue manager attribute, SCMDSERV, is specified as QMGR then changing the state of the command server using **strmqcsv** does not effect how the queue manager acts upon the SCMDSERV attribute at the next restart.

Syntax



Required parameters

None

Optional parameters

-a Blocks the following PCF commands from modifying or displaying authority information:

- Inquire authority records (**MQCMD_INQUIRE_AUTH_RECS**)
- Inquire entity authority (**MQCMD_INQUIRE_ENTITY_AUTH**)
- Set authority record (**MQCMD_SET_AUTH_REC**).
- Delete authority record (**MQCMD_DELETE_AUTH_REC**).

QMgrName
The name of the queue manager on which to start the command server. If omitted, the default queue manager is used.

Return codes

Return code	Description
0	Command completed normally
10	Command completed with unexpected results
20	An error occurred during processing

Examples

The following command starts a command server for queue manager `earth`:

```
strmqcsv earth
```

Related commands

Command	Description
endmqcsv	End a command server
dspmqcsv	Display the status of a command server

strmqsvc (Start IBM IBM WebSphere MQ service):

The **strmqsvc** command starts the IBM IBM WebSphere MQ service on Windows. Run the command on Windows only.

Purpose

The command starts the IBM IBM WebSphere MQ service on Windows.

Run the command to start the service, if it has not been started automatically, or if the service has ended.

Restart the service for IBM WebSphere MQ processes to pick up a new environment, including new security definitions.

Syntax

strmqsvc

Parameters

The **strmqsvc** command has no parameters.

You must set the path to the installation that contains the service. Either make the installation primary, run the **setmqenv** command, or run the command from the directory containing the **strmqsvc** binary file.

Related reference:

"endmqsvc (end IBM IBM WebSphere MQ service)" on page 192
The **endmqsvc** command ends the IBM IBM WebSphere MQ service on Windows. Run the command on Windows only.

strmqm:

Start a queue manager or ready it for standby operation.

Purpose

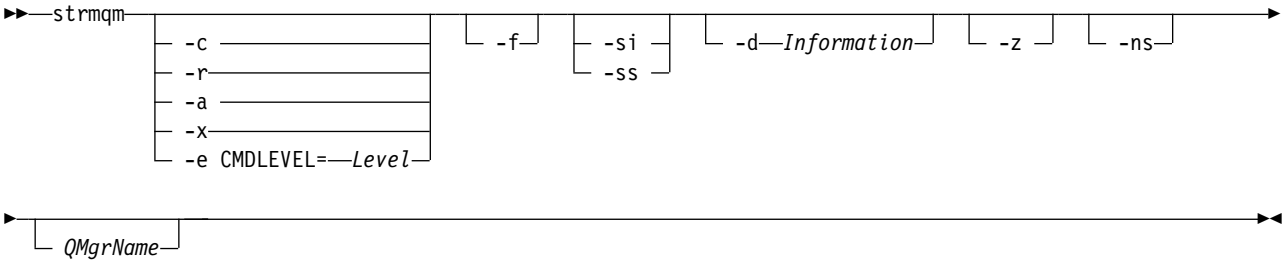
Use the **strmqm** command to start a queue manager.

You must use the **strmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

If a queue manager has no associated installation and there is no installation of IBM WebSphere MQ Version 7.0.1 on the system, the **strmqm** command will associate the queue manager with the installation that issued the **strmqm** command.

If the queue manager startup takes more than a few seconds IBM WebSphere MQ shows intermittent messages detailing the startup progress.

Syntax



Optional parameters

-a Activate the specified backup queue manager. The backup queue manager is not started.

When activated, a backup queue manager can be started using the control command `strmqm` *QMgrName*. The requirement to activate a backup queue manager prevents accidental startup.

When activated, a backup queue manager can no longer be updated.

For more information about using backup queue managers, see Backing up and restoring IBM WebSphere MQ queue manager data.

-c Starts the queue manager, redefines the default and system objects, then stops the queue manager. Any existing system and default objects belonging to the queue manager are replaced if you specify this flag, and any non-default system object values are reset (for example, the value of MCAUSER is set to blank).

Use the **crtmqm** command to create the default and system objects for a queue manager.

-d *Information*
Specifies whether information messages are displayed. Possible values for *Information* follow:

all	All information messages are displayed. This parameter is the default value.
minimal	The minimal number of information messages are displayed.
none	No information messages are displayed. This parameter is equivalent to **-z**.

The -*z* parameter takes precedence over this parameter.

-e CMDLEVEL=*Level*
Enables a command level for this queue manager, and then stops the queue manager.

The queue manager is now able to use all functions provided by the specified command level. You can start the queue manager only with an installation that supports the new command level.

This option is only valid if the current command level used by the queue manager is lower than the maximum command level supported by the installation. Specify a command level that is greater than the current command level of the queue manager and less than or equal to the maximum command level supported by the installation.

Use exactly the command level as a value for *Level* that is associated with the function you want to enable.

This flag cannot be specified with -a, -c, -r or -x.

-f Use this option if you *know* a queue manager is not starting because its data directories are missing or corrupted.

The **strmqm** `-f` *qmname* command attempts to re-create the queue manager data directory and reset file permissions. If it is successful, the queue manager starts, unless the queue manager configuration information is missing. If the queue manager fails to start because the configuration information is missing, re-create the configuration information, and restart the queue manager.

Before IBM WebSphere MQ Version 7.0.1, **strmqm**, with no `-f` option, automatically repaired missing data directories and then tried to start. This behavior has changed.

From IBM WebSphere MQ Version 7.0.1 onwards, the default behavior of **strmqm**, with no `-f` option, is *not* to recover missing or corrupted data directories automatically, but to report an error, such as AMQ6235 or AMQ7001, and *not* start the queue manager.

You can think of the `-f` option as performing the recover actions that used to be performed automatically by **strmqm**.

The reason for the change to the behavior of **strmqm** is that with the support for networked file storage in IBM WebSphere MQ Version 7.0.1, the most likely cause of missing or corrupted queue manager data directories is a configuration error that can be rectified, rather than the data directories are corrupted or irretrievably unavailable.

You must *not* use **strmqm** `-f` to re-create the queue manager data directories if you can restore the directories by correcting the configuration.

Possible solutions to problems with **strmqm** are to make the networked file storage location accessible to the queue manager, or to ensure the gid and uid of the mqm group and user ID on the server hosting the queue manager match the gid and uid of the mqm group and user ID on the server hosting the queue manager data directory.

From IBM WebSphere MQ Version 7.0.1, if you are performing media recovery for a queue manager, then you must use the `-f` option to re-create the queue manager data directory.

-ns
Prevents any of the following processes from starting automatically when the queue manager starts:

- The channel initiator
- The command server
- Listeners
- Services

-r Updates the backup queue manager. The backup queue manager is not started.

WebSphere MQ updates the objects of the backup queue manager by reading the queue manager log and replaying updates to the object files.

For more information about using backup queue managers, see Backing up and restoring IBM WebSphere MQ queue manager data.

-si
Interactive (manual) queue manager startup type. This option is available on IBM WebSphere MQ for Windows only.

The queue manager runs under the logged on (interactive) user. Queue managers configured with interactive startup end when the user who started them logs off.

If you set this parameter, it overrides any startup type set previously by the **crtmqm** command, the **amqmdain** command, or the IBM WebSphere MQ Explorer.

If you do not specify a startup type of either `-si` or `-ss`, the queue manager startup type specified on the **crtmqm** command is used.

-ss
Service (manual) queue manager startup type. This option is available on IBM WebSphere MQ for Windows only.

The queue manager runs as a service. Queue managers configured with service startup continue to run even after the interactive user has logged off.

If you set this parameter, it overrides any startup type set previously by the **crtmqm** command, the **amqmdain** command, or the IBM WebSphere MQ Explorer.

-x

Start an instance of a multi-instance queue manager on the local server, permitting it to be highly available. If an instance of the queue manager is not already running elsewhere, the queue manager starts and the instance becomes active. The active instance is ready to accept local and remote connections to the queue manager on the local server.

If a multi-instance queue manager instance is already active on a *different* server the new instance becomes a standby, permitting it to takeover from the active queue manager instance. While it is in standby, it cannot accept local or remote connections.

You must not start a second instance of a queue manager on the *same* server.

The default behavior, omitting the **-x** optional parameter, is to start the instance as a single instance queue manager, forbidding standby instances from being started.

-z Suppresses error messages.

This flag is used within IBM WebSphere MQ to suppress unwanted information messages. Because using this flag can result in loss of information, do not use it when entering commands on a command line.

This parameter takes precedence over the **-d** parameter.

QMgrName
The name of a local queue manager. If omitted, the default queue manager is used.

Return codes

Return code	Description
0	Queue manager started
3	Queue manager being created
5	Queue manager running
16	Queue manager does not exist
23	Log not available
24	A process that was using the previous instance of the queue manager has not yet disconnected.
30	A standby instance of the queue manager started. The active instance is running elsewhere
31	The queue manager already has an active instance. The queue manager permits standby instances.
39	Invalid parameter specified
43	The queue manager already has an active instance. The queue manager does not permit standby instances.
47	The queue manager already has the maximum number of standby instances
49	Queue manager stopping
58	Inconsistent use of installations detected
62	The queue manager is associated with a different installation
69	Storage not available
71	Unexpected error
72	Queue manager name error
74	The WebSphere MQ service is not started.
91	The command level is outside the range of acceptable values.
92	The queue manager's command level is greater or equal to the specified value.
100	Log location invalid
119	User not authorized to start the queue manager

Examples

The following command starts the queue manager `account`:

```
strmqm account
```

Related commands

Command	Description
"**crtmqm**" on page 146	Create a queue manager
"dltmqm" on page 153	Delete a queue manager
"dspmqver" on page 184	Display MQ version information
"endmqm" on page 189	End a queue manager

strmqtrc:

Enable trace at a specified level of detail, or report the level of tracing in effect.

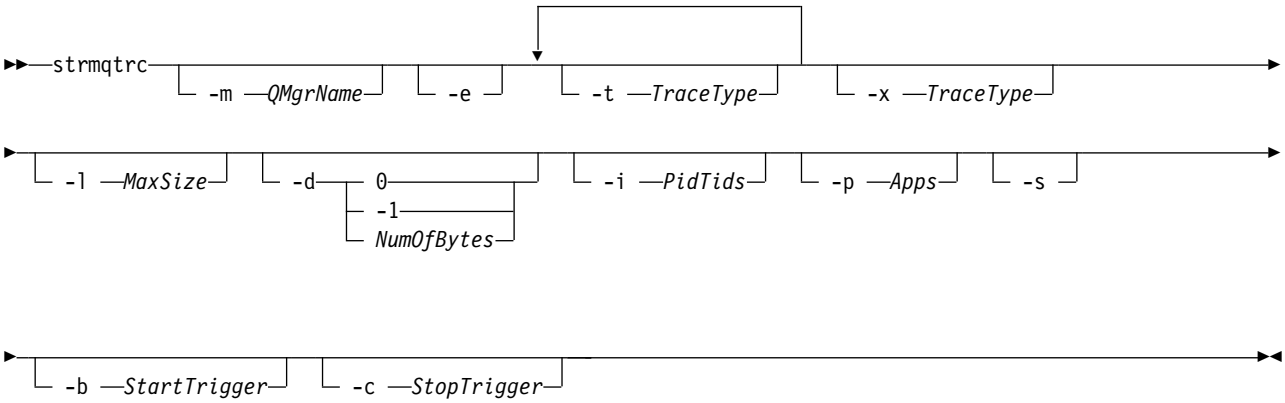
Purpose

Use the **strmqtrc** command to enable tracing.

You must use the **strmqtrc** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command. This does not apply to a client product (for example, HP Integrity NonStop Server) because there are no queue managers from which to request direct output.

Syntax

The syntax of this command is as follows:



Description

The strmqtrc command enables tracing. The command has optional parameters that specify the level of tracing you want:

- One or more queue managers
- Levels of trace detail
- One or more WebSphere MQ processes. The processes can be either part of the WebSphere MQ product or customer applications that use the WebSphere MQ API
- Specific threads within customer applications, either by WebSphere MQ thread number or by operating system thread number
- Events. These can be either the entry or exit from internal WebSphere MQ functions or the occurrence of a first failure data capture (FDC).

Each combination of parameters on an individual invocation of the command are interpreted by WebSphere MQ as having a logical AND between them. You can start the strmqtrc command multiple times, regardless of whether tracing is already enabled. If tracing is already enabled, the trace options that are in effect are modified to those specified on the most recent invocation of the command. Multiple invocations of the command, without an intervening enqmqtrc command, are interpreted by WebSphere MQ as having a logical OR between them. The maximum number of concurrent strmqtrc commands that can be in effect at one time is 16.

For the IBM WebSphere MQ client on HP Integrity NonStop Server, you must direct your trace commands to specific processors. For example, if your client is running on processor 2 and your shell is on processor 1, initiating trace with **strmqtrc <options>** does not trace the client. In this case, `run -cpu=2 strmqtrc` is required.

Optional parameters

-m *QMgrName*
The name of the queue manager to trace. This parameter applies to server products only.

The following wildcards are allowed: asterisk (*), replacing zero or more characters, and question mark (?), replacing any single character. In command environments such as the UNIX shell, where the asterisk (*) and question mark (?) characters have special meaning, you must either escape the wildcard character or enclose it in quotation marks to prevent the command environment from operating on the wildcard character.

-e Requests early tracing of all processes, making it possible to trace the creation or startup of a queue manager. If you include this flag, any process belonging to any component of any queue manager traces its early processing. The default is not to perform early tracing.

Use the following command to trace a client:

```
strmqtrc -e
```

You cannot use the -e flag with the -m flag, -i flag, the -p flag, the -c flag, or the -b flag. If you try to use the -e flag with the -m flag, the -i flag, the -p flag, the -c flag, or the -b flag, then an error message is issued.

-t *TraceType*
The points to trace and the amount of trace detail to record. By default **all** trace points are enabled and a default-detail trace is generated.

Alternatively, you can supply one or more of the options in the following list. For each *tracetype* value you specify, including `-t all`, specify either `-t parms` or `-t detail` to obtain the appropriate level of trace detail. If you do not specify either `-t parms` or `-t detail` for any particular trace type, only a default-detail trace is generated for that trace type.

If you supply multiple trace types, each must have its own -t flag. You can include any number of -t flags, if each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

all	Output data for every trace point in the system (the default). The all parameter activates tracing at default detail level.
api	Output data for trace points associated with the MQI and major queue manager components.
commentary	Output data for trace points associated with comments in the WebSphere MQ components.
comms	Output data for trace points associated with data flowing over communications networks.
csdata	Output data for trace points associated with internal data buffers in common services.

csflows	Output data for trace points associated with processing flow in common services.
detail	Activate tracing at high-detail level for flow processing trace points.
Explorer	Output data for trace points associated with the WebSphere MQ Explorer.
java	Output data for trace points associated with applications using the WebSphere MQ classes for Java API.
lqmdata	Output data for trace points associated with internal data buffers in the local queue manager.
lqmflows	Output data for trace points associated with processing flow in the local queue manager.
otherdata	Output data for trace points associated with internal data buffers in other components.
otherflows	Output data for trace points associated with processing flow in other components.
parms	Activate tracing at default-detail level for flow processing trace points.
remotedata	Output data for trace points associated with internal data buffers in the communications component.
remoteflows	Output data for trace points associated with processing flow in the communications component.
servicedata	Output data for trace points associated with internal data buffers in the service component.
serviceflows	Output data for trace points associated with processing flow in the service component.
spldata	Output data for trace points associated with buffers and control blocks that use a security policy (AMS) operation.
splflows	Output data for trace points associated with entry and exit data for functions that use a security policy (AMS) operation.
soap	Output data for trace points associated with WebSphere MQ Transport for SOAP.
ssl	Output data associated with using GSKit to enable Secure Sockets Layer (SSL) channel security.
versiondata	Output data for trace points associated with the version of WebSphere MQ running.

-x *TraceType*

The points **not** to trace. By default **all** trace points are enabled and a default-detail trace is generated. The trace points you can specify are those listed for the -t flag.

You can use the **-x** flag with *tracetype* values to exclude those entry points you do not want to record. This is useful in reducing the amount of trace produced.

If you supply multiple trace types, each must have its own -x flag. You can include any number of -x flags, if each has a valid *tracetype* associated with it.

-l *MaxSize*

The maximum size of a trace file (AMQ*ppppp*.*qq*.TRC) in megabytes (MB). For example, if you specify a MaxSize of 1, the size of the trace is limited to 1 MB.

When a trace file reaches the specified maximum, it is renamed to AMQ*ppppp*.*qq*.TRS and a new AMQ*ppppp*.*qq*.TRC file is started. If a previous copy of an AMQ*ppppp*.*qq*.TRS file exists, it is deleted.

The highest value that *MaxSize* can be set to is 2048 MB.

-d 0

Trace no user data.

-d -1 or all
Trace all user data.

-d *NumOfBytes*

- For a communication trace; trace the specified number of bytes of data including the transmission segment header (TSH).
- For an MQPUT or MQGET call; trace the specified number of bytes of message data held in the message buffer.
- Values in the range 1 through 15 are not allowed.

-i *PidTids*
Process identifier (PID) and thread identifier (TID) to which the trace generation is restricted. You cannot use the -i flag with the -e flag. If you try to use the -i flag with the -e flag, then an error message is issued. This parameter must only be used under the guidance of IBM Service personnel.

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed WebSphere MQ problem diagnostics.

-p *Apps*
The named processes to which the trace generation is restricted. *Apps* is a comma-separated list. You must specify each name in the list exactly as the program name would be displayed in the "Program Name" FDC header. Asterisk (*) or question mark (?) wildcards are allowed. You cannot use the -p flag with the -e flag. If you try to use the -p flag with the -e flag, then an error message is issued.

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed WebSphere MQ problem diagnostics.

-s
Reports the tracing options that are currently in effect. You must use this parameter on its own with no other parameters.

A limited number of slots are available for storing trace commands. When all slots are in use, then no more trace commands can be accepted unless they replace an existing slot. Slot numbers are not fixed, so if the command in slot number 0 is removed, for example by an endmqtrc command, then all the other slots move up, with slot 1 becoming slot 0, for example. An asterisk (*) in a field means that no value is defined, and is equivalent to the asterisk wildcard.

An example of the output from this command is as follows:

```
Listing Trace Control Array

Used slots = 2 of 15

EarlyTrace    [OFF]
TimedTrace    [OFF]
TraceUserData [0]
MaxSize       [0]
Trace Type    [1]

Slot position 1

Untriggered
Queue Manager [avocet]
Application   [*]
PID.TID       [*]
TraceOptions  [1f4ffff]
TraceInterval    [0]
Trace Start Time [0]
Trace Stop  Time [0]
Start Trigger [KN346050K]
Start Trigger [KN346080]

Slot position 2
```

```
Untriggered
Queue Manager [*]
Application   [*]
PID.TID       [*]
TraceOptions  [1fcffff]
TraceInterval    [0]
Trace Start Time [0]
Trace Stop  Time [0]
Start Trigger [KN346050K]
Start Trigger [KN346080]
```

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed WebSphere MQ problem diagnostics.

-b *Start_Trigger*

FDC probe IDs for which tracing must be turned on. *Start_Trigger* is a comma-separated list of FDC probe IDs. You can use asterisk (*) and question mark (?) wildcards in the specification of probe IDs. You cannot use the -b flag with the -e flag. If you try to use the -b flag with the -e flag, then an error message is issued. This parameter must only be used under the guidance of IBM Service personnel.

Start_Trigger	**Effect**
FDC=comma-separated list of FDC probe IDs.	Turns on tracing when any FDCs with the specified FDC probe IDs are generated.

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed WebSphere MQ problem diagnostics.

-c *Stop_Trigger*

FDC probe IDs for which tracing must be turned off, or interval in seconds after which tracing must be turned off. *Stop_Trigger* is a comma-separated list of FDC probe IDs. You can use asterisk (*) and question mark (?) wildcards in the specification of probe IDs. This parameter should be used only under the guidance of IBM Service personnel.

Stop_Trigger	**Effect**
FDC=comma-separated list of FDC probe IDs.	Turns tracing off when any FDCs with the specified FDC probe IDs are generated.
interval=n where n is an unsigned integer between 1 and 32,000,000.	Turns tracing off n seconds after it starts or, if it tracing is already enabled, turns tracing off n seconds after this instance of the command is issued.

This parameter is not supported for .NET clients if NMQ_MQ_LIB is set to `managed`, so that the client uses managed WebSphere MQ problem diagnostics.

Return codes

Return code	**Description**
AMQ7024	Non-valid arguments supplied to the command.
AMQ7077	You are not authorized to perform the requested operation.
AMQ8304	Nine concurrent traces (the maximum) already running.
58	Inconsistent use of installations detected

Examples

This command enables tracing of processing flow from common services and the local queue manager for a queue manager called QM1 in WebSphere MQ for UNIX systems. Trace data is generated at the default level of detail.

```
strmqtrc -m QM1 -t csflows -t lqmflows -t parms
```

This command disables tracing of SSL activity on a queue manager called QM1. Other trace data is generated at the parms level of detail.

```
strmqtrc -m QM1 -x ssl -t parms
```

This command enables high-detail tracing of the processing flow for all components:

```
strmqtrc -t all -t detail
```

This command enables tracing when FDC KN346050 or FDC KN346080 occur on any process that is using queue manager QM1:

```
strmqtrc -m QM1 -b FDC=KN346050,KN346080
```

This command enables tracing when FDC KN34650 occurs, and stops tracing when FDC KN346080 occurs. In both cases the FDC must occur on a process that is using queue manager QM1:

```
strmqtrc -m QM1 -b FDC=KN346050 -c FDC=KN346080
```

The next examples use the -p and -m flags to show the following:

- How a combination of parameters on an individual invocation of the command are interpreted by WebSphere MQ as having a logical AND between them.
- How multiple invocations of the command, without an intervening enqmqtrc command, are interpreted by WebSphere MQ as having a logical OR between them:

1. This command enables tracing for all threads that result from any executing process called amqxxx.exe:

   ```
   strmqtrc -p amqxxx.exe
   ```

2.
   - If you start the following command after the command in step 1, without an intervening endmqtrc command, then tracing is limited to all threads that result from any executing process called amqxxx.exe *and* that are using queue manager QM2:

     ```
     strmqtrc -p amqxxx.exe -m QM2
     ```

   - If you start the following command after the command in step 1, without an intervening endmqtrc command, then tracing is limited to all processes and threads that result from executing amqxxx.exe *or* that are using queue manager QM2:

     ```
     strmqtrc -m QM2
     ```

Related commands

Command	Description
dspmqtrc	Display formatted trace output
endmqtrc	End trace

Comparing command sets

The tables in this section compare the facilities available from the different administration command sets, and also show whether you can perform each function from within the IBM WebSphere MQ Explorer.

Note: The following tables do not apply to IBM WebSphere MQ for z/OS or IBM WebSphere MQ for IBM i.

Queue manager commands:

A table of queue manager commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 50. Queue manager commands

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Queue Manager	Change Queue Manager	ALTER QMGR	No equivalent	Yes
Create Queue Manager	No equivalent	No equivalent	**crtmqm**	Yes
Delete Queue Manager	No equivalent	No equivalent	**dltmqm**	Yes
Inquire Queue Manager	Inquire Queue Manager	DISPLAY QMGR	No equivalent	Yes
Inquire Queue Manager Status	Inquire Queue Manager Status	DISPLAY QMSTATUS	**dspmq**	Yes
Ping Queue Manager	Ping Queue Manager	PING QMGR	No equivalent	No
Refresh Queue Manager	No equivalent	REFRESH QMGR	No equivalent	No
Reset Queue Manager	Reset Queue Manager	RESET QMGR	No equivalent	No
Start Queue Manager	No equivalent	No equivalent	**strmqm**	Yes
Stop Queue Manager	No equivalent	No equivalent	**endmqm**	Yes

Command server commands:

A table of command server commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 51. Commands for command server administration

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Display command server	Inquire Queue Manager Status	DISPLAY QMSTATUS	**dspmqcsv**	Yes
Start command server	Change Queue Manager	ALTER QMGR	**strmqcsv**	Yes
Stop command server	No equivalent	No equivalent	**endmqcsv**	Yes

Authority commands:

A table of authority commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 52. Commands for authority administration

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Delete authority record	DELETE AUTHREC	**setmqaut**	Yes
Inquire authority records	DISPLAY AUTHREC	**dmpmqaut**	Yes
Inquire entity authority	DISPLAY ENTAUTH	**dspmqaut**	Yes
Refresh Security	REFRESH SECURITY	No equivalent	Yes
Set authority record	SET AUTHREC	**setmqaut**	Yes

Cluster commands:

A table of cluster commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 53. Cluster commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Inquire Cluster Queue Manager	DISPLAY CLUSQMGR	No equivalent	Yes
Refresh Cluster	REFRESH CLUSTER	No equivalent	Yes
Reset Cluster	RESET CLUSTER	No equivalent	No
Resume Queue Manager Cluster	RESUME QMGR	No equivalent	Yes
Suspend Queue Manager Cluster	SUSPEND QMGR	No equivalent	Yes

Authentication information commands:

A table of authentication information commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 54. Authentication information commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Authentication Information Object	ALTER AUTHINFO	No equivalent	Yes
Copy Authentication Information Object	DEFINE AUTHINFO(x) LIKE(y)	No equivalent	Yes
Create Authentication Information Object	DEFINE AUTHINFO	No equivalent	Yes
Delete Authentication Information Object	DELETE AUTHINFO	No equivalent	Yes
Inquire Authentication Information Object	DISPLAY AUTHINFO	No equivalent	Yes

Channel commands:

A table of channel commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and IBM WebSphere MQ Explorer equivalents, if available.

Table 55. Channel commands

PCF command	MQSC command	Control command	IBM WebSphere MQ Explorer equivalent?
Change Channel	ALTER CHANNEL	No equivalent	Yes
Copy Channel	DEFINE CHANNEL(x) LIKE(y)	No equivalent	Yes
Create Channel	DEFINE CHANNEL	No equivalent	Yes
Delete Channel	DELETE CHANNEL	No equivalent	Yes
Inquire Channel	DISPLAY CHANNEL	No equivalent	Yes
Inquire Channel Names	DISPLAY CHANNEL	No equivalent	Yes
Inquire Channel Status	DISPLAY CHSTATUS	No equivalent	Yes
Ping Channel	PING CHANNEL	No equivalent	Yes
Purge Channel	PURGE CHANNEL	No equivalent	Yes
Reset Channel	RESET CHANNEL	No equivalent	Yes
Resolve Channel	RESOLVE CHANNEL	No equivalent	Yes
Start Channel	START CHANNEL	**runmqchl**	Yes
Start Channel Initiator	START CHINIT	**runmqchi**	No
Stop Channel	STOP CHANNEL	No equivalent	Yes

Listener commands:

A table of listener commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 56. Listener commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Listener	ALTER LISTENER	No equivalent	Yes
Copy Listener	DEFINE LISTENER(x) LIKE(y)	No equivalent	Yes
Create Listener	DEFINE LISTENER	No equivalent	Yes
Delete Listener	DELETE LISTENER	No equivalent	Yes
Inquire Listener	DISPLAY LISTENER	No equivalent	Yes
Inquire Listener Status	DISPLAY LSSTATUS	No equivalent	Yes
Start Channel Listener	START LISTENER [1]	**runmqlsr**	Yes
Stop Listener	STOP LISTENER	**endmqlsr** [2]	Yes
Notes: 1. Used with listener objects only 2. Stops all active listeners			

Namelist commands:

A table of namelist commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 57. Namelist commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Namelist	ALTER NAMELIST	No equivalent	Yes
Copy Namelist	DEFINE NAMELIST(x) LIKE(y)	No equivalent	Yes
Create Namelist	DEFINE NAMELIST	No equivalent	Yes
Delete Namelist	DELETE NAMELIST	No equivalent	Yes
Inquire Namelist	DISPLAY NAMELIST	No equivalent	Yes
Inquire Namelist Names	DISPLAY NAMELIST	No equivalent	Yes

Process commands:

A table of process commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 58. Process commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Process	ALTER PROCESS	No equivalent	Yes
Copy Process	DEFINE PROCESS(x) LIKE(y)	No equivalent	Yes
Create Process	DEFINE PROCESS	No equivalent	Yes
Delete Process	DELETE PROCESS	No equivalent	Yes
Inquire Process	DISPLAY PROCESS	No equivalent	Yes
Inquire Process Names	DISPLAY PROCESS	No equivalent	Yes

Queue commands:

A table of queue commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 59. Queue commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Queue	ALTER QLOCAL ALTER QALIAS ALTER QMODEL ALTER QREMOTE	No equivalent	Yes
Clear Queue	CLEAR QLOCAL	No equivalent	Yes

Table 59. Queue commands (continued)

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Copy Queue	DEFINE QLOCAL(x) LIKE(y) DEFINE QALIAS(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)	No equivalent	Yes
Create Queue	DEFINE QLOCAL DEFINE QALIAS DEFINE QMODEL DEFINE QREMOTE	No equivalent	Yes
Delete Queue	DELETE QLOCAL DELETE QALIAS DELETE QMODEL DELETE QREMOTE	No equivalent	Yes
Inquire Queue	DISPLAY QUEUE	No equivalent	Yes
Inquire Queue Names	DISPLAY QUEUE	No equivalent	Yes
Inquire Queue Status	DISPLAY QSTATUS	No equivalent	Yes
Reset Queue Statistics	No equivalent	No equivalent	No

Service commands:

A table of service commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 60. Service commands

PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Change Service	ALTER SERVICE	No equivalent	Yes
Copy Service	DEFINE SERVICE(x) LIKE(y)	No equivalent	Yes
Create Service	DEFINE SERVICE	No equivalent	Yes
Delete Service	DELETE SERVICE	No equivalent	Yes
Inquire Service	DISPLAY SERVICE	No equivalent	Yes
Inquire Service Status	DISPLAY SVSTATUS	No equivalent	Yes
Start Service	START SERVICE	No equivalent	Yes
Stop Service	STOP SERVICE	No equivalent	Yes

Other commands:

A table of other commands, showing the command description, and its PCF command, MQSC command, control command equivalents, and WebSphere MQ Explorer equivalents, if available.

Table 61. Other commands

Description	PCF command	MQSC command	Control command	WebSphere MQ Explorer equivalent?
Create conversion exit	No equivalent	No equivalent	**crtmqcvx**	No
Display files used by objects	No equivalent	No equivalent	**dspmqfls**	No
Display formatted trace	No equivalent	No equivalent	**dspmqtrc** [1]	No
Display version information	No equivalent	No equivalent	**dspmqver**	No
Display transactions	No equivalent	No equivalent	**dspmqtrn**	No
Dump log	No equivalent	No equivalent	**dmpmqlog**	No
Dump MQ Configuration	No equivalent	No equivalent	**dmpmqcfg**	No
End trace	No equivalent	No equivalent	**endmqtrc**	Yes
Escape	Escape	No equivalent	No equivalent	No
Record media image	No equivalent	No equivalent	**rcdmqimg**	No
Re-create media object	No equivalent	No equivalent	**rcrmqobj**	No
Resolve transactions	No equivalent	No equivalent	**rsvmqtrn**	No
Run client trigger monitor	No equivalent	No equivalent	**runmqtmc**	No
Run dead-letter queue handler	No equivalent	No equivalent	**runmqdlq**	No
Run MQSC commands	No equivalent	No equivalent	**runmqsc**	No
Run trigger monitor	No equivalent	No equivalent	**runmqtrm**	No
Set service connection points	No equivalent	No equivalent	**setmqscp** [2]	No
Start WebSphere MQ trace	No equivalent	No equivalent	**strmqtrc**	Yes
WebSphere MQ Services control	No equivalent	No equivalent	**amqmdain** [2]	No

Notes:

1. Not supported on WebSphere MQ for Windows.
2. Supported by WebSphere MQ for Windows only.

Managing keys and certificates

Use the **runmqckm** command (Windows and UNIX systems) to manage keys, certificates, and certificate requests.

The runmqckm command

The **runmqckm** command is available on Windows and UNIX systems.

The **runmqckm** command provides functions that are similar to those of iKeyman, described in Security.

Use the **runmqckm** command to do the following:

- Create the type of CMS key database files that WebSphere MQ requires
- Create certificate requests
- Import personal certificates
- Import CA certificates
- Manage self-signed certificates

Preparing to use the runmqckm and runmqakm commands:

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

To run the **runmqckm** command line interfaces, ensure that the systems environment variables are correctly configured. For primary installations of WebSphere MQ v7.1, you can run the **setmqinst** command. For further information on this command please see “setmqinst” on page 234

runmqckm, and runmqakm commands:

This section describes the runmqckm, and runmqakm commands according to the object of the command.

An overview of the main differences between the two commands:

- **runmqakm**
  - Supports the creation of certificates and certificate requests with Elliptic Curve public keys whereas the **runmqckm** command does not.
  - Supports stronger encryption of the key repository file than the **runmqckm** command through the **-strong** parameter.
  - Has been certified as FIPS 140-2 compliant, and can be configured to operate in a FIPS compliant manner, using the **-fips** parameter, unlike the **runmqckm** command.
- **runmqckm** supports the JKS and JCEKS key repository file formats whereas the **runmqakm** command does not.

Each command specifies at least one *object*. Commands for PKCS #11 device operations might specify additional objects. Commands for key database, certificate, and certificate request objects also specify an *action*. The object can be one of the following:

-keydb
Actions apply to a key database

-cert Actions apply to a certificate

-certreq
Actions apply to a certificate request

-help Displays help

-version
Displays version information

The following subtopics describe the actions that you can take on key database, certificate, and certificate request objects; See "runmqckm and runmqakm options" on page 268 for a description of the options on these commands.

Commands for a CMS key database only:

You can use the **runmqckm**, and **runmqakm** commands to manage keys and certificates for a CMS key database.

-keydb -changepw
Change the password for a CMS key database:

```
-keydb -changepw -db filename -pw password -new_pw new_password

-stash
```

-keydb -create
Create a CMS key database:

```
-keydb -create -db filename -pw password -type cms -expire days -stash
```

-keydb -stashpw
Stash the password of a CMS key database into a file:

```
-keydb -stashpw -db filename -pw password
```

-cert -getdefault
Get the default personal certificate:

```
-cert -getdefault -db filename -pw password
```

-cert -modify
Modify a certificate.

Note: Currently, the only field that can be modified is the Certificate Trust field.

```
-cert -modify -db filename -pw password -label label
    -trust enable | disable
```

-cert -setdefault
Set the default personal certificate:

```
-cert -setdefault -db filename -pw password -label label
```

Command for CMS or PKCS #12 key databases:

You can use the runmqckm, and runmqakm commands to manage keys and certificates for a CMS key database or PKCS #12 key database.

Note: WebSphere MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.

The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.

-keydb -changepw
Change the password for a key database:

```
-keydb -changepw -db filename -pw password -new_pw new_password -expire days
```

-keydb -convert

convert the key database from one format to another:

```
-keydb -convert -db filename -pw password
    -old_format cms | pkcs12 -new_format cms
```

-keydb -create

Create a key database:

```
-keydb -create -db filename -pw password -type cms
 | pkcs12
```

-keydb -delete

Delete a key database:

```
-keydb -delete -db filename -pw password
```

-keydb -list

List currently-supported types of key database:

```
-keydb -list
```

-cert -add

Add a certificate from a file into a key database:

```
-cert -add -db filename -pw password -label label
 -file filename
    -format ascii | binary
```

-cert -create

Create a self-signed certificate:

```
-cert -create -db filename -pw password -label label
 -dn distinguished_name
    -size 1024 | 512 -x509version 3 | 1
 | 2
    -expire days -sig_alg MD2_WITH_RSA | MD2WithRSA
 |
                              MD5_WITH_RSA | MD5WithRSA
 |
                              SHA1WithDSA | SHA1WithRSA
 |
                              SHA256_WITH_RSA | SHA256WithRSA
 |
                              SHA2WithRSA | SHA384_WITH_RSA
 |
                              SHA384WithRSA |  SHA512_WITH_RSA
 |
                              SHA512WithRSA | SHA_WITH_DSA
 |
                              SHA_WITH_RSA | SHAWithDSA
 |
                              SHAWithRSA
```

-cert -delete

Delete a certificate:

```
-cert -delete -db filename -pw password -label label
```

-cert -details

List the detailed information for a specific certificate:

```
-cert -details -db filename -pw password -label label
```

-cert -export

Export a personal certificate and its associated private key from a key database into a PKCS #12 file, or to another key database:

```
-cert -export -db filename -pw password -label label
 -type cms | pkcs12
    -target filename -target_pw password -target_type cms | pkcs12
```

-cert -extract

Extract a certificate from a key database:

```
-cert -extract -db filename -pw password -label label
 -target filename
    -format ascii | binary
```

-cert -import

Import a personal certificate from a key database:

```
 -cert -import -file filename -pw password -type pkcs12 -target filename
       -target_pw password -target_type cms -label label
```

The -label option is required and specifies the label of the certificate that is to be imported from the source key database.

The -new_label option is optional and allows the imported certificate to be given a different label in the target key database from the label in the source database.

-cert -list

List all certificates in a key database:

```
-cert -list all | personal | CA
    -db filename -pw password
```

-cert -receive

Receive a certificate from a file:

```
-cert -receive -file filename -db filename -pw password

    -format ascii | binary -default_cert yes | no
```

-cert -sign

Sign a certificate:

```
-cert -sign -db filename -file filename  -pw password
   -label label -target filename
   -format ascii | binary -expire days
   -sig_alg MD2_WITH_RSA | MD2WithRSA | MD5_WITH_RSA
 |
           MD5WithRSA | SHA1WithDSA | SHA1WithRSA
 |
           SHA256_WITH_RSA | SHA256WithRSA |
           SHA2WithRSA | SHA384_WITH_RSA |
           SHA384WithRSA |  SHA512_WITH_RSA |
           SHA512WithRSA | SHA_WITH_DSA  |
           SHA_WITH_RSA | SHAWithDSA |
           SHAWithRSA
```

-certreq -create

Create a certificate request:

```
-certreq -create -db filename -pw password
    -label label -dn distinguished_name
    -size 1024 | 512 -file filename
    -sig_alg MD2_WITH_RSA | MD2WithRSA |
             MD5_WITH_RSA | MD5WithRSA |
             SHA1WithDSA | SHA1WithRSA |
             SHA256_WITH_RSA | SHA256WithRSA |
             SHA2WithRSA | SHA384_WITH_RSA |
             SHA384WithRSA |  SHA512_WITH_RSA |
             SHA512WithRSA | SHA_WITH_DSA  |
             SHA_WITH_RSA | SHAWithDSA |
             SHAWithRSA
```

-certreq -delete

Delete a certificate request:

```
-certreq -delete -db filename -pw password -label label
```

-certreq -details

List the detailed information of a specific certificate request:

```
-certreq -details -db filename -pw password -label label
```

List the detailed information about a certificate request and show the full certificate request:

```
-certreq -details -showOID -db filename
    -pw password -label label
```

-certreq -extract

Extract a certificate request from a certificate request database into a file:

```
-certreq -extract -db filename -pw password
     -label label -target filename
```

-certreq -list

List all certificate requests in the certificate request database:

```
-certreq -list -db filename -pw password
```

-certreq -recreate

Re-create a certificate request:

```
-certreq -recreate -db filename -pw password
    -label label -target filename
```

Commands for cryptographic device operations:

You can use the runmqckm, and runmqakm commands to manage keys and certificates for cryptographic device operations.

Note: WebSphere MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.

The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.

-keydb -changepw

Change the password for a cryptographic device:

```
-keydb -changepw -crypto module_name -tokenlabel token_label
    -pw password -new_pw new_password
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-keydb -list

List currently-supported types of key database:

```
-keydb -list
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-cert -add

Add a certificate from a file to a cryptographic device:

```
-cert -add -crypto module_name -tokenlabel token_label
    -pw password -label label -file filename -format ascii | binary
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-cert -create

Create a self-signed certificate on a cryptographic device:

```
-cert -create -crypto module_name -tokenlabel token_label

    -pw password -label label -dn distinguished_name
 -size 1024 | 512
    -x509version 3 | 1 | 2 -default_cert no
 | yes -expire days
    -sig_alg MD2_WITH_RSA | MD2WithRSA |
             MD5_WITH_RSA | MD5WithRSA |
             SHA1WithDSA | SHA1WithRSA |
             SHA256_WITH_RSA | SHA256WithRSA |
             SHA2WithRSA | SHA384_WITH_RSA |
             SHA384WithRSA | SHA512_WITH_RSA |
             SHA512WithRSA | SHA_WITH_DSA  |
             SHA_WITH_RSA | SHAWithDSA |
             SHAWithRSA
```

Note: You cannot import a certificate containing multiple OU (organizational unit) attributes in the distinguished name.

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-cert -delete

Delete a certificate on a cryptographic device:

```
-cert -delete -crypto module_name -tokenlabel token_label
    -pw password -label label
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-cert -details

List the detailed information for a specific certificate on a cryptographic device:

```
-cert -details -crypto module_name -tokenlabel token_label

    -pw password -label label
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

List the detailed information and show the full certificate for a specific certificate on a cryptographic device:

```
-cert -details -showOID -crypto module_name -tokenlabel token_label
    -pw password -label label
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-cert -extract

Extract a certificate from a key database:

```
-cert -extract -crypto module_name -tokenlabel token_label
    -pw password -label label -target filename
 -format ascii | binary
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-cert -import

Import a certificate to a cryptographic device with secondary key database support:

```
-cert -import -db filename -pw password -label label
 -type cms
    -crypto module_name -tokenlabel token_label -pw password
    -secondaryDB filename -secondaryDBpw password
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

```
-cert -import -db filename -pw password -label label
 -type cms
    -crypto module_name -tokenlabel token_label -pw password
    -secondaryDB filename -secondaryDBpw password -fips
```

Import a PKCS #12 certificate to a cryptographic device with secondary key database support:

```
-cert -import -file filename -pw password -type pkcs12
    -crypto module_name -tokenlabel token_label -pw password
    -secondaryDB filename -secondaryDBpw password
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

```
-cert -import -file filename -pw password -type pkcs12
    -crypto module_name -tokenlabel token_label -pw password
    -secondaryDB filename -secondaryDBpw password -fips
```

Note: You cannot import a certificate containing multiple OU (organizational unit) attributes in the distinguished name.

-cert -list

List all certificates on a cryptographic device:

```
-cert -list all | personal | CA
    -crypto module_name -tokenlabel token_label -pw password
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-cert -receive

Receive a certificate from a file to a cryptographic device with secondary key database support:

```
-cert -receive -file filename -crypto module_name -tokenlabel token_label
    -pw password  -default_cert yes | no
    -secondaryDB filename -secondaryDBpw password -format ascii | binary
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

Using the **runmqakm** command:

-certreq -create

Create a certificate request on a cryptographic device:

```
-certreq -create -crypto module_name -tokenlabel token_label

    -pw password -label label -dn distinguished_name
    -size 1024 | 512 -file filename
    -sig_alg MD2_WITH_RSA | MD2WithRSA |MD5_WITH_RSA
 |
             MD5WithRSA |SHA1WithDSA | SHA1WithRSA
 |
             SHA256_WITH_RSA | SHA256WithRSA
             SHA2WithRSA | SHA384_WITH_RSA |
             SHA384WithRSA | SHA512_WITH_RSA |
             SHA512WithRSA | SHA_WITH_DSA  |
             SHA_WITH_RSA | SHAWithDSA |
             SHAWithRSA
```

Note: You cannot import a certificate containing multiple OU (organizational unit) attributes in the distinguished name.

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-certreq -delete

Delete a certificate request from a cryptographic device:

```
-certreq -delete -crypto module_name -tokenlabel token_label

    -pw password -label label
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-certreq -details

List the detailed information of a specific certificate request on a cryptographic device:

```
-certreq -details -crypto module_name -tokenlabel token_label

    -pw password -label label
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

List the detailed information about a certificate request and show the full certificate request on a cryptographic device:

```
-certreq -details -showOID -crypto module_name -tokenlabel token_label
    -pw password -label label
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-certreq -extract

Extract a certificate request from a certificate request database on a cryptographic device into a file:

```
-certreq -extract -crypto module_name -tokenlabel token_label

    -pw password -label label -target filename
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

-certreq -list

List all certificate requests in the certificate request database on a cryptographic device:

```
-certreq -list -crypto module_name -tokenlabel token_label

    -pw password
```

If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 64-bit programs. External modules required for PKCS #11 support will be loaded into a 64-bit process, therefore you must have a 64-bit PKCS #11 library installed for the administration of cryptographic hardware. The Windows and Linux x86 32-bit platforms are the only exceptions, as the iKeyman and iKeycmd programs are 32-bit on those platforms.

runmqckm and runmqakm options:

A table of the runmqckm and runmqakm options that can be present on the command line.

Note: WebSphere MQ does not support SHA-3 or SHA-5 algorithms. You can use the digital signature algorithm names SHA384WithRSA and SHA512WithRSA because both algorithms are members of the SHA-2 family.

The digital signature algorithm names SHA3WithRSA and SHA5WithRSA are deprecated because they are an abbreviated form of SHA384WithRSA and SHA512WithRSA respectively.

The meaning of an option can depend on the object and action specified in the command.

Table 62. Options that can be used with **runmqckm** *and* **runmqakm**

Option	Description
`-create`	Option to create a key database.
`-crypto`	Name of the module to manage a PKCS #11 cryptographic device. The value after `-crypto` is optional if you specify the module name in the properties file.If you are using certificates or keys stored on PKCS #11 cryptographic hardware, note that iKeycmd and iKeyman are 32-bit programs. External modules required for PKCS #11 support will be loaded into a 32-bit process, therefore you must have a 32-bit PKCS #11 library installed for the administration of cryptographic hardware, and must specify this library to iKeycmd or iKeyman. The HP Itanium platform is the only exception, as the iKeyman program is 64-bit on the HP Itanium platform.
`-db`	Fully qualified path name of a key database.
`-default_cert`	Sets a certificate as the default certificate. The value can be `yes` or `no`. The default is `no`.
`-dn`	X.500 distinguished name. The value is a string enclosed in double quotation marks, for example "`CN=John Smith,O=IBM,OU=Test,C=GB`". Note that the CN, O, and C attributes are required. **Note:** Avoid using multiple OU attributes in distinguished names when you create self-signed certificates. When you create such certificates, only the **last entered** OU value is accepted into the certificate.
`-encryption`	Strength of encryption used in certificate export command. The value can be `strong` or `weak` . The default is `strong`.
`-expire`	Expiration time in days of either a certificate or a database password. The default is 365 days for a certificate password. There is no default time for a database password: use the `-expire` option to set a database password expiration time explicitly.
`-file`	File name of a certificate or certificate request.
`-format`	Format of a certificate. The value can be `ascii` for Base64_encoded ASCII or `binary` for Binary DER data. The default is `ascii`.
`-label`	Label attached to a certificate or certificate request.
`-new_format`	New format of key database.
`-new_label`	Used on a certificate import command, this option allows a certificate to be imported with a different label from the label it had in the source key database.
`-new_pw`	New database password.
`-old_format`	Old format of key database.
`-pw`	Password for the key database or PKCS #12 file.
`-secondaryDB`	Name of a secondary key database for PKCS #11 device operations.
`-secondaryDBpw`	Password for the secondary key database for PKCS #11 device operations.

Table 62. Options that can be used with **runmqckm** *and* **runmqakm** *(continued)*

Option	Description
-showOID	Displays the full certificate or certificate request.
-sig_alg	The hashing algorithm used during the creation of a certificate request, a self-signed certificate, or the signing of a certificate. This hashing algorithm is used to create the signature associated with the newly-created certificate or certificate request. For runmqckm, the value can be MD2_WITH_RSA, MD2WithRSA, MD5_WITH_RSA, MD5WithRSA , SHA1WithDSA, SHA1WithRSA, SHA256_WITH_RSA, SHA256WithRSA, SHA2WithRSA, SHA384_WITH_RSA, SHA384WithRSA , SHA512_WITH_RSA, SHA512WithRSA , SHA_WITH_DSA, SHA_WITH_RSA, SHAWithDSA, or SHAWithRSA. The default value is SHA1WithRSA. For runmqakm, the value can be md5, MD5_WITH_RSA, MD5WithRSA, SHA_WITH_DSA, SHA_WITH_RSA , sha1, SHA1WithDSA, SHA1WithECDSA , SHA1WithRSA, sha224, SHA224_WITH_RSA, SHA224WithDSA, SHA224WithECDSA, SHA224WithRSA, sha256, SHA256_WITH_RSA, SHA256WithDSA , SHA256WithECDSA, SHA256WithRSA , SHA2WithRSA, sha384, SHA384_WITH_RSA, SHA384WithECDSA, SHA384WithRSA, sha512, SHA512_WITH_RSA , SHA512WithECDSA, SHA512WithRSA , SHAWithDSA, SHAWithRSA, EC_ecdsa_with_SHA1, EC_ecdsa_with_SHA224 , EC_ecdsa_with_SHA256, EC_ecdsa_with_SHA384 , or EC_ecdsa_with_SHA512. The default value is SHA1WithRSA.
-size	Key size. For runmqckm, the value can be 512, 1024, or 2048. The default value is 1024 bits. For runmqakm, the value depends upon the signature algorithm: • For RSA signature algorithms (the default algorithm used if no -sig_alg is specified), the value can be 512, 1024, 2048, or 4096. An RSA key size of 512 bits os not permitted if the **-fips** parameter is enabled. The default RSA key size is 1024 bits. • For Elliptic Curve algorithms, the value can be 256, 384, or 512. The default Elliptic Curve key size depends upon the signature algorithm. For SHA256, it is 256; for SHA384 , it is 384; and for SHA512, it is 512.
-stash	Stash the key database password to a file.
-target	Destination file or database.
-target_pw	Password for the key database if -target specifies a key database.
-target_type	Type of database specified by -target operand. See -type option for permitted values.
-tokenLabel	Label of a PKCS #11 cryptographic device.
-trust	Trust status of a CA certificate. The value can be enable or disable. The default is enable.
-type	Type of database. The value can be: • cms for a CMS key database • pkcs12 for a PKCS #12 file.
-x509version	Version of X.509 certificate to create. The value can be 1, 2, or 3. The default is 3.

Note: Properties provided with IBM Global Secure Toolkit (GSKit) relating to symmetric-key encryption -seckey option in the 'runmqckm' utility are ignored and not supported by WebSphere MQ.

runmqakm error codes:

A table of the numeric error codes issued by runmqakm, and what they mean.

Error code	Error Message
0	Success
1	Unknown error occurred
2	An ASN.1 encoding/decoding error occurred.
3	An error occurred while initializing ASN.1 encoder/decoder.
4	An ASN.1 encoding/decoding error occurred because of an out-of-range index or non-existent optional field.
5	A database error occurred.
6	An error occurred while opening the database file, check for file existence and permission.
7	An error occurred while re-opening the database file.
8	Database creation failed.
9	The database already exists.
10	An error occurred while deleting the database file.
11	The database could not be opened.
12	An error occurred while reading the database file.
13	An error occurred while writing data to the database file.
14	A database validation error occurred.
15	An invalid database version was encountered.
16	An invalid database password was encountered.
17	An invalid database file type was encountered.
18	The specified database has been corrupted.
19	An invalid password was provided or the key database has been tampered with or corrupted.
20	A database key entry integrity error occurred.
21	A duplicate certificate already exists in the database.
22	A duplicate key already exists in the database (Record ID).
23	A certificate with the same label already existed in the key database.
24	A duplicate key already exists in the database (Signature).
25	A duplicate key already exists in the database (Unsigned Certificate).
26	A duplicate key already exists in the database (Issuer and Serial Number).
27	A duplicate key already exists in the database (Subject Public Key Info).
28	A duplicate key already exists in the database (Unsigned CRL).
29	The label has been used in the database.

Error code	Error Message
30	A password encryption error occurred.
31	An LDAP related error occurred. (LDAP is not supported by this program)
32	A cryptographic error occurred.
33	An encryption/decryption error occurred.
34	An invalid cryptographic algorithm was found.
35	An error occurred while signing data.
36	An error occurred while verifying data.
37	An error occurred while computing digest of data.
38	An invalid cryptographic parameter was found.
39	An unsupported cryptographic algorithm was encountered.
40	The specified input size is greater than the supported modulus size.
41	An unsupported modulus size was found.
42	A database validation error occurred.
43	Key entry validation failed.
44	A duplicate extension field exists.
45	The version of the key is wrong.
46	A required extension field does not exist.
47	The validity period does not include today or does not fall within its issuer's validity period
48	The validity period does not include today or does not fall within its issuer's validity period.
49	An error occurred while validating private key usage extension.
50	The issuer of the key was not found.
51	A required certificate extension is missing.
52	An invalid basic constraint extension was found.
53	The key signature validation failed.
54	The root key of the key is not trusted.
55	The key has been revoked.
56	An error occurred while validating authority key identifier extension.
57	An error occurred while validating private key usage extension.
58	An error occurred while validating subject alternative name extension.
59	An error occurred while validating issuer alternative name extension.
60	An error occurred while validating key usage extension.
61	An unknown critical extension was found.
62	An error occurred while validating key pair entries.
63	An error occurred while validating CRL.

Error code	Error Message
64	A mutex error occurred.
65	An invalid parameter was found.
66	A null parameter or memory allocation error was encountered.
67	Number or size is too large or too small.
68	The old password is invalid.
69	The new password is invalid.
70	The password has expired.
71	A thread related error occurred.
72	An error occurred while creating threads.
73	An error occurred while a thread was waiting to exit.
74	An I/O error occurred.
75	An error occurred while loading CMS.
76	A cryptography hardware related error occurred.
77	The library initialization routine was not successfully called.
78	The internal database handle table is corrupted.
79	A memory allocation error occurred.
80	An unrecognized option was found.
81	An error occurred while getting time information.
82	Mutex creation error occurred.
83	An error occurred while opening message catalog.
84	An error occurred while opening error message catalog
85	A null file name was found.
86	An error occurred while opening files, check for file existence and permissions.
87	An error occurred while opening files to read.
88	An error occurred while opening files to write.
89	There is no such file.
90	The file cannot be opened because of its permission setting.
91	An error occurred while writing data to files.
92	An error occurred while deleting files.
93	Invalid Base64-encoded data was found.
94	An invalid Base64 message type was found.
95	An error occurred while encoding data with Base64 encoding rule.
96	An error occurred while decoding Base64-encoded data.
97	An error occurred while getting a distinguished name tag.
98	The required common name field is empty.
99	The required country or region name field is empty.

Error code	Error Message
100	An invalid database handle was found.
101	The key database does not exist.
102	The request key pair database does not exist.
103	The password file does not exist.
104	The new password is identical to the old one.
105	No key was found in the key database.
106	No request key was found.
107	No trusted CA was found.
108	No request key was found for the certificate.
109	There is no private key in the key database.
110	There is no default key in the key database.
111	There is no private key in the key record.
112	There is no certificate in the key record.
113	There is no CRL entry.
114	An invalid key database file name was found.
115	An unrecognized private key type was found.
116	An invalid distinguished name input was found.
117	No key entry was found that has the specified key label.
118	The key label list has been corrupted.
119	The input data is not valid PKCS12 data.
120	The password is invalid or the PKCS12 data has been corrupted or been created with later version of PKCS12
121	An unrecognized key export type was found.
122	An unsupported password-based encryption algorithm was found.
123	An error occurred while converting the key ring file to a CMS key database.
124	An error occurred while converting the CMS key database to a key ring file.
125	An error occurred while creating a certificate for the certificate request.
126	A complete issuer chain cannot be built.
127	Invalid WEBDB data was found.
128	There is no data to be written to the key ring file.
129	The number of days that you entered extends beyond the permitted validity period.
130	The password is too short; it must consist of at least {0} characters.
131	A password must contain at least one numeric digit.
132	All characters in the password are either alphabetic or numeric characters.
133	An unrecognized or unsupported signature algorithm was specified.

Error code	Error Message
134	An invalid database type was encountered.
135	The specified secondary key database is in use by another PKCS#11 device.
136	No secondary key database was specified.
137	The label does not exist on the PKCS#11 device.
138	Password required to access the PKCS#11 device.
139	Password not required to access the PKCS#11 device.
140	Unable to load the cryptographic library.
141	PKCS#11 is not supported for this operation.
142	An operation on a PKCS#11 device has failed.
143	The LDAP user is not a valid user. (LDAP is not supported by this program)
144	The LDAP user is not a valid user. (LDAP is not supported by this program)
145	The LDAP query failed. (LDAP is not supported by this program)
146	An invalid certificate chain was found.
147	The root certificate is not trusted.
148	A revoked certificate was encountered.
149	A cryptographic object function failed.
150	There is no certificate revocation list data source available.
151	There is no cryptographic token available.
152	FIPS mode is not available.
153	There is a conflict with the FIPS mode settings.
154	The password entered does not meet the minimum required strength.
200	There was a failure during initialization of the program.
201	Tokenization of the arguments passed to the runmqakm Program failed.
202	The object identified in the command is not a recognized object.
203	The action passed is not a known -keydb action.
204	The action passed is not a known -cert action.
205	The action passed is not a known -certreq action.
206	There is a tag missing for the requested command.
207	The value passed with the -version tag is not a recognized value.
208	The value passed with the -size tag is not a recognized value.
209	The value passed in with the -dn tag is not in the correct format.
210	The value passed in with the -format tag is not a recognized value.

Error code	Error Message
211	There was an error associated with opening the file.
212	PKCS12 is not supported at this stage.
213	The cryptographic token you are trying to change the password for is not password protected.
214	PKCS12 is not supported at this stage.
215	The password entered does not meet the minimum required strength.
216	FIPS mode is not available.
217	The number of days you have entered as the expiry date is out of the allowed range.
218	Password strength failed the minimum requirements.
219	No Default certificate was found in the requested key database.
220	An invalid trust status was encountered.
221	An unsupported signature algorithm was encountered. At this stage only MD5 and SHA1 are supported.
222	PCKS11 not supported for that particular operation.
223	The action passed is not a known -random action.
224	A length than less than zero is not allowed.
225	When using the -strong tag the minimum length password is 14 characters.
226	When using the -strong tag the maximum length password is 300 characters.
227	The MD5 algorithm is not supported when in FIPS mode.
228	The site tag is not supported for the -cert -list command. This attribute is added for backward compatibility and potential future enhancement.
229	The value associated with the -ca tag is not recognized. The value must be either 'true' or 'false'.
230	The value passed in with the -type tag is not valid.
231	The value passed in with the -expire tag is below the allowed range.
232	The encryption algorithm used or requested is not supported.
233	The target already exists.

MQSC reference

Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

For an overview of using MQSC commands for administering IBM WebSphere MQ, see Performing local administration tasks using MQSC commands.

MQSC commands use certain special characters to have certain meanings. For more information about these special characters and how to use them, see "Generic values and characters with special meanings."

To find out how you can build scripts using MQSC commands, see "Building command scripts" on page 277.

For the full list of MQSC commands, see "The MQSC commands" on page 278.

Related concepts:

"WebSphere MQ Control commands" on page 131
Find out how to use the WebSphere MQ control commands.

"Programmable command formats reference" on page 796
Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. PCFs simplify queue manager administration and other network administration.

Generic values and characters with special meanings

The following information describes generic values, and characters that have special meaning when you build MQSC commands.

Wherever a parameter can have a generic value, it is entered ending with an asterisk (*), for example ABC*. A generic value means 'all values beginning with'; so ABC* means 'all values beginning with ABC'.

If characters that require quotation marks are used in the value, the asterisk must be placed inside the quotation marks, thus 'abc*'. The asterisk must be the last or only character in the value.

The question mark (?) and colon (:) are not allowed in generic values.

Character	Description
	Blanks are used as separators. Multiple blanks are equivalent to a single blank, except in strings that have apostrophes (') round them. Any trailing blanks in those string attributes which are based on MQCHARV types are treated as significant.
,	Commas are used as separators. Multiple commas are equivalent to a single comma, except in strings that have apostrophes (') round them.
'	An apostrophe indicates the beginning or end of a string. IBM WebSphere MQ leaves all characters that have quotation marks round them exactly as they are entered. The containing apostrophes are not included when calculating the length of the string.
''	Single quotation marks inside a string are treated by IBM WebSphere MQ as one character when calculating the length of the string and the string is not terminated.
=	On z/OS, an equals sign indicates the start of a parameter value which is ended by a comma or blank.
(	An open parenthesis indicates the beginning of a parameter value or list of values.
)	A close parenthesis indicates the end of a parameter value or list of values.
:	A colon indicates an inclusive range. For example (1:5) means (1,2,3,4,5). This notation can be used only in TRACE commands.

Character	Description
*	An asterisk means "all". For example, DISPLAY TRACE (*) means display all traces, and DISPLAY QUEUE (PAY*) means display all queues with names that begin with PAY.

When you need to use any of these special characters in a field (for example as part of a description), you must enclose the whole string in single quotation marks.

Building command scripts

Use this information to learn how to build command scripts.

You might want to build the MQSC commands into a script when you use:

- The CSQINP1, CSQINP2, and CSQINPX initialization data sets or the CSQUTIL batch utility on z/OS.
- The STRMQM command on IBM i.
- The **runmqsc** command on UNIX, Linux, and Windows systems.

When you do this, follow these rules:

- Each command must start on a new line.
- On each platform, there might be platform-specific rules about the line length and record format. If scripts are to be readily portable to different platforms, the significant length of each line should be restricted to 72 characters.
  - On z/OS, scripts are held in a fixed-format data set, with a record length of 80. Only columns 1 through 72 can contain meaningful information; columns 73 through 80 are ignored.
  - On AIX, HP-UX, Linux, IBM i, Solaris, and Windows, each line can be of any length up to a maximum of 2048 characters.
  - On other UNIX systems, each line can be of any length up to and including 80 characters.
- A line must not end in a keyboard control character (for example, a tab).
- If the last nonblank character on a line is:
  - A minus sign (-), this indicates that the command is to be continued from the start of the next line.
  - A plus sign (+), this indicates that the command is to be continued from the first nonblank character in the next line. If you use + to continue a command remember to leave at least one blank before the next parameter (except on z/OS where this is not necessary).

  Either of these can occur within a parameter, data value, or a string enclosed in quotation marks. For example,

  ```
  'Fr+
   ed'
  ```

  and

  ```
  'Fr-
ed'
  ```

  (where the 'e' of the second line of the second example is in the first position of the line) are both equivalent to

  ```
  'Fred'
  ```

  MQSC commands that are contained within an Escape PCF (Programmable Command Format) command cannot be continued in this way. The entire command must be contained within a single Escape command. (For information about the PCF commands, see Introduction to Programmable Command Formats).
- + and - values used at the ends of lines are discarded when the command is reassembled into a single string.

- On AIX, HP-UX, Linux, IBM i, SolarisSolaris, and Windows you can use a semicolon character (;) to terminate a command, even if you have entered a plus sign (+) at the end of the previous line. You can also use the semicolon in the same way on z/OS for commands issued from the CSQUTIL batch utility program.
- A line starting with an asterisk (*) in the first position is ignored. This can be used to insert comments into the file.

  A blank line is also ignored.

  If a line ends with a continuation character (- or +), the command continues with the next line that is not a comment line or a blank line.
- When running MQSC commands interactively, you end the interactive session by typing the END command. This applies to:
  - UNIX, Linux, and Windows systems, where you start the interactive session by entering **runmqsc**
  - IBM i systems, where you start the interactive session from the WRKMQM panel
- On Windows, if certain special characters such as the pound sign (£) and the logical NOT (¬) are used in a command script (for example, as part of an object description), they will be displayed differently in the output from a command such as DISPLAY QLOCAL.

The MQSC commands

Use this topic as a reference to the MQSC commands.

This section describes, in alphabetical order, all the MQSC commands that can be issued by operators and administrators.

"START CHANNEL" on page 779
"START CHANNEL (MQTT)" on page 782
"START CHINIT" on page 782

"START LISTENER" on page 783

"START SERVICE" on page 785

"STOP CHANNEL" on page 786
"STOP CHANNEL (MQTT)" on page 790

"STOP CONN" on page 790
"STOP LISTENER" on page 791

"STOP SERVICE" on page 793

"SUSPEND QMGR" on page 794

Related information:
Clustering: Using REFRESH CLUSTER best practices

ALTER AUTHINFO:

Use the MQSC command ALTER AUTHINFO to alter an authentication information object.

These objects contain the definitions required to perform certificate revocation checking using OCSP or Certificate Revocation Lists (CRLs) on LDAP servers.

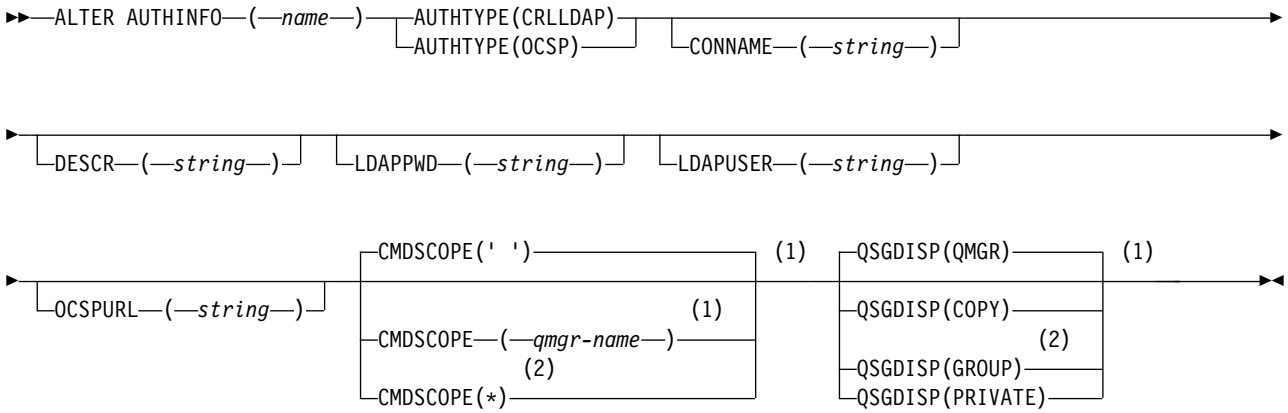
UNIX and Linux	Windows
✓	✓

Parameters not specified in the ALTER AUTHINFO command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Parameter descriptions for ALTER AUTHINFO" on page 282

Synonym: ALT AUTHINFO

ALTER AUTHINFO



Notes:

1 Valid only on z/OS.

2 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on WebSphere MQ for z/OS.

Parameter descriptions for ALTER AUTHINFO

name Name of the authentication information object. This parameter is required.

The name must not be the same as any other authentication information object name currently defined on this queue manager (unless REPLACE or ALTER is specified). See Rules for naming IBM WebSphere MQ objects.

AUTHTYPE
The type of authentication information.

CRLLDAP
Certificate Revocation List checking is done using LDAP servers.

OCSP
Certificate revocation checking is done using OCSP.

An authentication information object with AUTHTYPE(OCSP) does not apply for use on IBM i or z/OS queue managers. However, it can be specified on those platforms to be copied to the client channel definition table (CCDT) for client use.

This parameter is required.

You cannot define an authentication information object as LIKE one with a different AUTHTYPE. You cannot alter the AUTHTYPE of an authentication information object after you have created it.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

'' The command is executed on the queue manager on which it was entered.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

CONNAME(*string***)**
The host name, IPv4 dotted decimal address, or IPv6 hexadecimal notation of the host on which the LDAP server is running, with an optional port number.

CONNAME is required if AUTHTYPE(CRLLDAP) is specified. CONNAME is not valid if AUTHTYPE(CRLLDAP) is not specified.

If you specify the connection name as an IPv6 address, only systems with an IPv6 stack are able to resolve this address. If the AUTHINFO object is part of the CRL namelist of the queue manager, ensure that any clients using the client channel table generated by the queue manager can resolve the connection name.

On z/OS, if a CONNAME is to resolve to an IPv6 network address, a level of z/OS that supports IPv6 for connection to an LDAP server is required.

The syntax for CONNAME is the same as for channels. For example,

```
conname('hostname(nnn)')
```

where *nnn* is the port number.

The maximum length for the field is 264 characters on IBM i, UNIX systems, and Windows, and 48 characters on z/OS.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the authentication information object when an operator issues the DISPLAY AUTHINFO command (see "DISPLAY AUTHINFO" on page 578).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LDAPPWD(*string***)**
The password associated with the Distinguished Name of the user who is accessing the LDAP server. Its maximum size is 32 characters.

This parameter is valid only for AUTHTYPE(CRLLDAP).

On z/OS, the LDAPPWD used for accessing the LDAP server might not be the one defined in the AUTHINFO object. If more than one AUTHINFO object is placed in the namelist referred to by the QMGR parameter SSLCRLNL, the LDAPPWD in the first AUTHINFO object is used for accessing all LDAP Servers.

LDAPUSER(*string***)**
The Distinguished Name of the user who is accessing the LDAP server. (See the SSLPEER parameter for more information about distinguished names.)

This parameter is valid only for AUTHTYPE(CRLLDAP).

The maximum size for the user name is 1024 characters on IBM i, UNIX systems, and Windows, and 256 characters on z/OS.

On z/OS, the LDAPUSER used for accessing the LDAP Server might not be the one defined in the AUTHINFO object. If more than one AUTHINFO object is placed in the namelist referred to by the QMGR parameter SSLCRLNL, the LDAPUSER in the first AUTHINFO object is used for accessing all LDAP Servers.

On IBM i, UNIX systems, and Windows, the maximum accepted line length is defined to be BUFSIZ, which can be found in stdio.h.

OCSPURL

The URL of the OCSP responder used to check for certificate revocation. This value must be an HTTP URL containing the host name and port number of the OCSP responder. If the OCSP responder is using port 80, which is the default for HTTP, then the port number can be omitted. HTTP URLs are defined in RFC 1738.

This field is case sensitive. It must start with the string http:// in lowercase. The rest of the URL might be case sensitive, depending on the OCSP server implementation. To preserve case, use single quotation marks to specify the OCSPURL parameter value, for example:

```
OCSPURL('http://ocsp.example.ibm.com')
```

This parameter is applicable only for AUTHTYPE(OCSP), when it is mandatory.

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero:
	`DEFINE AUTHINFO(name)`
	`REPLACE QSGDISP(COPY)`
	
	The ALTER for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with QSGDISP(QMGR) or QSGDISP(COPY). Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

ALTER CHANNEL:

Use the MQSC command ALTER CHANNEL to alter the parameters of a channel.

UNIX and Linux	Windows
✓	✓

Parameters not specified in the ALTER CHANNEL command result in the existing values for those parameters being left unchanged.

Synonym: ALT CHL

- "Usage notes"
- "Parameter descriptions for ALTER CHANNEL"

Usage notes

- Changes take effect after the channel is next started.
- For cluster-sender channels, you can only alter channels that have been created manually.
- If you change the XMITQ name or the CONNAME, you must reset the sequence number at both ends of the channel. (See "RESET CHANNEL" on page 757 for information about the SEQNUM parameter.)

Parameter descriptions for ALTER CHANNEL

The following table shows the parameters that are relevant for each type of channel. There is a description of each parameter after the table. Parameters are optional unless the description states that they are required.

Table 63. ALTER CHANNEL parameters

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR	MQTT
AFFINITY					✓				
BATCHHB	✓	✓					✓	✓	
BATCHINT	✓	✓					✓	✓	
BATCHLIM	✓	✓					✓	✓	
BATCHSZ	✓	✓	✓	✓			✓	✓	
channel-name	✓	✓	✓	✓	✓	✓	✓	✓	✓
CHLTYPE	✓	✓	✓	✓	✓	✓	✓	✓	✓
CLNTWGHT					✓				
CLUSNL							✓	✓	
CLUSTER							✓	✓	
CLWLPRTY							✓	✓	

Table 63. ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR	MQTT
CLWLRANK							✓	✓	
CLWLWGHT							✓	✓	
CMDSCOPE	✓	✓	✓	✓	✓	✓	✓	✓	
COMPHDR	✓	✓	✓	✓	✓	✓	✓	✓	
COMPMSG	✓	✓	✓	✓	✓	✓	✓	✓	
CONNAME	✓	✓		✓	✓		✓	✓	
CONVERT	✓	✓					✓	✓	
DEFCDISP	✓	✓	✓	✓		✓			
DEFRECON					✓				
DESCR	✓	✓	✓	✓	✓	✓	✓	✓	✓
DISCINT	✓	✓				✓	✓	✓	
HBINT	✓	✓	✓	✓	✓	✓	✓	✓	
KAINT	✓	✓	✓	✓	✓	✓	✓	✓	
LIKE	✓	✓	✓	✓	✓	✓	✓	✓	✓
LOCLADDR	✓	✓		✓	✓		✓	✓	✓
LONGRTY	✓	✓					✓	✓	
LONGTMR	✓	✓					✓	✓	
MAXINST						✓			
MAXINSTC						✓			
MAXMSGL	✓	✓	✓	✓	✓	✓	✓	✓	
MCANAME	✓	✓		✓			✓	✓	
MCATYPE	✓	✓		✓			✓	✓	
MCAUSER			✓	✓		✓		✓	✓
MODENAME	✓	✓		✓	✓		✓	✓	

Table 63. ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR	MQTT
MONCHL	✓	✓	✓	✓		✓	✓	✓	
MRDATA			✓	✓				✓	
MREXIT			✓	✓				✓	
MRRTY			✓	✓				✓	
MRTMR			✓	✓				✓	
MSGDATA	✓	✓	✓	✓			✓	✓	
MSGEXIT	✓	✓	✓	✓			✓	✓	
NETPRTY								✓	
NPMSPEED	✓	✓	✓	✓			✓	✓	
PASSWORD	✓	✓		✓	✓		✓	✓	
PROPCTL	✓	✓					✓	✓	
PUTAUT			✓	✓		✓		✓	
QMNAME					✓				
QSGDISP	✓	✓	✓	✓	✓	✓	✓	✓	
RCVDATA	✓	✓	✓	✓	✓	✓	✓	✓	
RCVEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
REPLACE	✓	✓	✓	✓	✓	✓	✓	✓	
SCYDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SCYEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SENDDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SENDEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SEQWRAP	✓	✓	✓	✓			✓	✓	
SHARECNV					✓	✓			
SHORTRTY	✓	✓					✓	✓	

Table 63. ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR	MQTT
SHORTTMR	✓	✓					✓	✓	
SSLCAUTH		✓	✓	✓		✓		✓	✓
SSLCIPH[1]	✓	✓	✓	✓	✓	✓	✓	✓	✓ [1]
SSLCIPH									✓
SSLPEER	✓	✓	✓	✓	✓	✓	✓	✓	
STATCHL	✓	✓	✓	✓			✓	✓	
TPNAME	✓	✓		✓	✓	✓	✓	✓	
TRPTYPE	✓	✓	✓	✓	✓	✓	✓	✓	✓
USEDLQ	✓	✓	✓	✓			✓	✓	
USERID	✓	✓		✓	✓		✓		
XMITQ	✓	✓							

Note:

1. If SSLCIPH is used with MQTT channels, it means SSL Cipher Suite. For all other channel types, it means SSL CipherSpec. See SSLCIPH.

AFFINITY

The channel affinity attribute is used so client applications that connect multiple times using the same queue manager name can choose whether to use the same client channel definition for each connection. This attribute is intended to be used when multiple applicable channel definitions are available.

PREFERRED

The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any applicable CLNTWGHT(0) definitions first and in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful non-CLNTWGHT(0) definitions are moved to the end of the list. CLNTWGHT(0) definitions remain at the start of the list and are selected first for each connection. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created. Each client process with the same host name creates the same list.

NONE

The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the weighting with any applicable CLNTWGHT(0) definitions selected first in alphabetical order. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created.

For example, suppose we had the following definitions in the CCDT:

```
CHLNAME(A) QMNAME(QM1) CLNTWGHT(3)
CHLNAME(B) QMNAME(QM1) CLNTWGHT(4)
CHLNAME(C) QMNAME(QM1) CLNTWGHT(4)
```

The first connection in a process creates its own ordered list based on the weightings. So it might, for example, create the ordered list CHLNAME(B), CHLNAME(A), CHLNAME(C).

For AFFINITY(PREFERRED), each connection in the process attempts to connect using CHLNAME(B). If a connection is unsuccessful the definition is moved to the end of the list which now becomes CHLNAME(A), CHLNAME(C), CHLNAME(B). Each connection in the process then attempts to connect using CHLNAME(A).

For AFFINITY(NONE), each connection in the process attempts to connect using one of the three definitions selected at random based on the weightings.

When sharing conversations is enabled with a non-zero channel weighting and AFFINITY(NONE), multiple connections in a process using the same queue manager name can connect using different applicable definitions rather than sharing an existing channel instance.

BATCHHB(*integer***)**
Specifies whether batch heartbeats are to be used. The value is the length of the heartbeat in milliseconds.

Batch heartbeats allow a sending channel to verify that the receiving channel is still active just before committing a batch of messages, so that if the receiving channel is not active, the batch can be backed out rather than becoming in-doubt, as would otherwise be the case. By backing out the batch, the messages remain available for processing so they could, for example, be redirected to another channel.

If the sending channel has had a communication from the receiving channel within the batch heartbeat interval, the receiving channel is assumed to be still active. If not, a 'heartbeat' is sent to the receiving channel to check.

The value must be in the range zero through 999999. A value of zero indicates that batch heartbeating is not used.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, and CLUSRCVR.

BATCHINT(*integer***)**
The minimum amount of time, in milliseconds, that a channel keeps a batch open.

The batch is terminated when one of the following conditions is met:

- BATCHSZ messages have been sent.
- BATCHLIM bytes have been sent.
- The transmission queue is empty and BATCHINT is exceeded.

The value must be in the range 0 - 999999999. Zero means that the batch is terminated as soon as the transmission queue becomes empty (or the BATCHSZ limit is reached).

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

BATCHLIM(*integer***)**

The limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point. A sync point is taken after the message that caused the limit to be reached has flowed across the channel. A value of zero in this attribute means that no data limit is applied to batches over this channel.

The batch is terminated when one of the following conditions is met:

- BATCHSZ messages have been sent.
- BATCHLIM bytes have been sent.

- The transmission queue is empty and BATCHINT is exceeded.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

The value must be in the range 0 - 999999. The default value is 5000.

This parameter is supported on all platforms.

BATCHSZ(*integer***)**
The maximum number of messages that can be sent through a channel before taking a sync point.

The maximum batch size used is the lowest of the following values:

- The BATCHSZ of the sending channel.
- The BATCHSZ of the receiving channel.
- On z/OS, three less than the maximum number of uncommitted messages allowed at the sending queue manager (or one if this value is zero or less). On platforms other than z/OS, the maximum number of uncommitted messages allowed at the sending queue manager (or one if this value is zero or less).
- On z/OS, three less than the maximum number of uncommitted messages allowed at the receiving queue manager (or one if this value is zero or less). On platforms other than z/OS, the maximum number of uncommitted messages allowed at the receiving queue manager (or one if this value is zero or less).

The maximum number of uncommitted messages is specified by the MAXUMSGS parameter of the ALTER QMGR command.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

The value must be in the range 1 through 9999.

(channel-name)
The name of the new channel definition.

This parameter is required on all types of channel. On CLUSSDR channels, it can take a different form from the other channel types. If your convention for naming cluster-sender channels includes the name of the queue manager, you can define a cluster-sender channel using the `+QMNAME+` construction. After connection to the matching cluster-receiver channel, IBM WebSphere MQ substitutes the correct repository queue manager name in place of `+QMNAME+` in the cluster-sender channel definition. This facility applies to AIX, HP-UX, Linux, IBM i, Solaris, and Windows only; for further information see Components of a cluster.

The name must not be the same as any existing channel defined on this queue manager (unless REPLACE or ALTER is specified). On z/OS, client-connection channel names can duplicate others.

The maximum length of the string is 20 characters, and the string must contain only valid characters; see Rules for naming IBM WebSphere MQ objects.

CHLTYPE
Channel type. This parameter is required. It must follow immediately after the *(channel-name)* parameter on all platforms except z/OS.

SDR Sender channel

SVR Server channel

RCVR Receiver channel

RQSTR Requester channel

CLNTCONN
Client-connection channel

SVRCONN
Server-connection channel

CLUSSDR
Cluster-sender channel

CLUSRCVR
Cluster-receiver channel

Note: If you are using the REPLACE option, you cannot change the channel type.

CLNTWGHT
The client channel weighting attribute is used so client channel definitions can be selected at random based on their weighting when more than one suitable definition is available. Specify a value in the range 0 - 99.

The special value 0 indicates that no random load balancing is performed and applicable definitions are selected in alphabetical order. To enable random load balancing the value can be in the range 1 through 99 where 1 is the lowest weighting and 99 is the highest.

When a client issues an MQCONN with queue manager name "*<name>" and more than one suitable definition is available in the CCDT the choice of definition to use is randomly selected based on the weighting with any applicable CLNTWGHT(0) definitions selected first in alphabetical order. The distribution is not guaranteed.

For example, suppose we had the following two definitions in the CCDT:

```
CHLNAME(TO.QM1) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address1) CLNTWGHT(2)
CHLNAME(TO.QM2) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address2) CLNTWGHT(4)
```

A client MQCONN with queue manager name "*GRP1" would choose one of the two definitions based on the weighting of the channel definition. (A random integer 1 - 6 would be generated. If the integer was in the range 1 through 2 address1 would be used otherwise address2 would be used). If this connection was unsuccessful the client would then use the other definition.

The CCDT might contain applicable definitions with both zero and non-zero weighting. In this situation, the definitions with zero weightings are chosen first and in alphabetical order. If these connections are unsuccessful the definitions with non-zero weighting are chosen based on their weighting.

For example, suppose we had the following four definitions in the CCDT:

```
CHLNAME(TO.QM1) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address1) CLNTWGHT(1)
CHLNAME(TO.QM2) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address2) CLNTWGHT(2)
CHLNAME(TO.QM3) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address3) CLNTWGHT(0)
CHLNAME(TO.QM4) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address4) CLNTWGHT(0)
```

A client MQCONN with queue manager name "*GRP1" would first choose definition "TO.QM3". If this connection was unsuccessful the client would then choose definition "TO.QM4". If this connection was also unsuccessful the client would then randomly choose one of the remaining two definitions based on their weighting.

CLNTWGHT support is added for all supported transport protocols.

CLUSNL(*nlname***)**
The name of the namelist that specifies a list of clusters to which the channel belongs.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLUSSDR and CLUSRCVR channels. Only one of the resultant values of CLUSTER or CLUSNL can be nonblank, the other must be blank.

CLUSTER(*clustername***)**
The name of the cluster to which the channel belongs. The maximum length is 48 characters conforming to the rules for naming IBM WebSphere MQ objects.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLUSSDR or CLUSRCVR. Only one of the resultant values of CLUSTER or CLUSNL can be nonblank, the other must be blank.

CLWLPRTY(*integer***)**
Specifies the priority of the channel for the purposes of cluster workload distribution. The value must be in the range zero through 9 where zero is the lowest priority and 9 is the highest.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLUSSDR or CLUSRCVR.

For more information about this attribute, see `CLWLPRTY` queue attribute.

CLWLRANK(*integer***)**
Specifies the rank of the channel for the purposes of cluster workload distribution. The value must be in the range zero through 9 where zero is the lowest rank and 9 is the highest.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLUSSDR or CLUSRCVR.

For more information about this attribute, see `CLWLRANK` channel attribute.

CLWLWGHT(*integer***)**
Specifies the weighting to be applied to the channel for the purposes of cluster workload distribution so that the proportion of messages sent down the channel can be controlled. The value must be in the range 1 through 99 where 1 is the lowest rank and 99 is the highest.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLUSSDR or CLUSRCVR.

For more information about this attribute, see CLWLWGHT channel attribute .

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

`' '` The command is executed on the queue manager on which it was entered.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

COMPHDR
The list of header data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used.

The mutually supported compression techniques of the channel are passed to the message exit of the sending channel where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

NONE No header data compression is performed.

SYSTEM Header data compression is performed.

COMPMSG

The list of message data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used.

The mutually supported compression techniques of the channel are passed to the message exit of the sending channel where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

NONE No message data compression is performed.

RLE Message data compression is performed using run-length encoding.

ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

ANY Any compression technique supported by the queue manager can be used. This value is only valid for receiver, requester, and server-connection channels.

CONNAME(*string***)**

Connection name.

For cluster-receiver channels (when specified) CONNAME relates to the local queue manager, and for other channels it relates to the target queue manager.

The maximum length of the string is 48 characters on z/OS, and 264 characters on other platforms.

A workaround to the 48 character limit might be one of the following suggestions:

- Set up your DNS servers so that you use, for example, host name of "myserver" instead of "myserver.location.company.com", ensuring you can use the short host name.
- Use IP addresses.

Specify CONNAME as a comma-separated list of names of machines for the stated TRPTYPE. Typically only one machine name is required. You can provide multiple machine names to configure multiple connections with the same properties. The connections are usually tried in the order they are specified in the connection list until a connection is successfully established. The order is modified for clients if the CLNTWGHT attribute is provided. If no connection is successful, the channel attempts the connection again, as determined by the attributes of the channel. With client channels, a connection-list provides an alternative to using queue manager groups to configure multiple connections. With message channels, a connection list is used to configure connections to the alternative addresses of a multi-instance queue manager.

This parameter is required for channels with a channel type (CHLTYPE) of SDR, RQSTR, CLNTCONN, and CLUSSDR. It is optional for SVR channels, and for CLUSRCVR channels of TRPTYPE(TCP), and is not valid for RCVR or SVRCONN channels.

Providing multiple connection names in a list was first supported in IBM WebSphere MQ Version 7.0.1. It changes the syntax of the CONNAME parameter. Earlier clients and queue managers connect using the first connection name in the list, and do not read the rest of the connection names in the list. In order for the earlier clients and queue managers to parse the new syntax, you must specify a port number on the first connection name in the list. Specifying a port number avoids problems when connecting to the channel from a client or queue manager that is running at a level earlier than IBM WebSphere MQ Version 7.0.1.

On AIX, HP-UX, IBM i, Linux, Solaris, and Windows platforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, WebSphere MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:

```
(1415)
```

The generated CONNAME is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

Note: If you are using any of the special characters in your connection name (for example, parentheses) you must enclose the string in single quotation marks.

The value you specify depends on the transport type (TRPTYPE) to be used:

LU 6.2

- On z/OS, there are two forms in which to specify the value:

  Logical unit name
  The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. Logical unit name can be specified in one of three forms:

Form	Example
luname	IGY12355
luname/TPname	IGY12345/APING
luname/TPname/modename	IGY12345/APINGD/#INTER

  For the first form, the TP name and mode name must be specified for the TPNAME and MODENAME parameters; otherwise these parameters must be blank.

  Note: For client-connection channels, only the first form is allowed.

  Symbolic name
  The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The TPNAME and MODENAME parameters must be blank.

  Note: For cluster-receiver channels, the side information is on the other queue managers in the cluster. Alternatively, in this case it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

  The specified or implied LU name can be that of a VTAM generic resources group.
- On IBM i, Windows, UNIX and Linux systems, CONNAME is the name of the CPI-C communications side object or, if the TPNAME is not blank, CONNAME is the fully qualified name of the partner logical unit.

NetBIOS
A unique NetBIOS name (limited to 16 characters).

SPX The 4 byte network address, the 6 byte node address, and the 2 byte socket number. These values must be entered in hexadecimal, with a period separating the network and node addresses. The socket number must be enclosed in brackets, for example:

```
CONNAME('0a0b0c0d.804abcde23a1(5e86)')
```

TCP Either the host name, or the network address of the remote machine (or the local machine for cluster-receiver channels). This address can be followed by an optional port number, enclosed in parentheses.

If the CONNAME is a host name, the host name is resolved to an IP address.

The IP stack used for communication depends on the value specified for CONNAME **and** the value specified for LOCLADDR. See LOCLADDR for information about how this value is resolved.

On z/OS, the connection name can include the IP_name of an z/OS dynamic DNS group or a Network Dispatcher input port. Do **not** include the IP_name or input port for channels with a channel type (CHLTYPE) of CLUSSDR.

When you define a channel with a channel type (CHLTYPE) of CLUSRCVR that is using TCP/IP, you do not need to specify the network address of your queue manager. IBM WebSphere MQ generates a CONNAME for you, assuming the default port and using the current IPv4 address of the system. If the system does not have an IPv4 address, the current IPv6 address of the system is used.

Note: If you are using clustering between IPv6-only and IPv4-only queue managers, do not specify an IPv6 network address as the CONNAME for CLUSRCVR channels. A queue manager that is capable only of IPv4 communication is unable to start a cluster sender channel definition that specifies the CONNAME in IPv6 hexadecimal form. Consider, instead, using host names in a heterogeneous IP environment.

CONVERT

Specifies whether the sending message channel agent attempts conversion of the application message data, if the receiving message channel agent cannot perform this conversion.

NO No conversion by sender

YES Conversion by sender

On z/OS, `N` and `Y` are accepted as synonyms of `NO` and `YES`.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

DEFCDISP

Specifies the default channel disposition of the channel.

PRIVATE
The intended disposition of the channel is as a PRIVATE channel.

FIXSHARED
The intended disposition of the channel is as a FIXSHARED channel.

SHARED
The intended disposition of the channel is as a SHARED channel.

This parameter does not apply to channels with a channel type (CHLTYPE) of CLNTCONN, CLUSSDR, or CLUSRCVR.

DEFRECON

Specifies whether a client connection automatically reconnects a client application if its connection breaks.

NO Unless overridden by MQCONNX, the client is not reconnected automatically.

YES Unless overridden by MQCONNX, the client reconnects automatically.

QMGR Unless overridden by MQCONNX, the client reconnects automatically, but only to the same queue manager. The `QMGR` option has the same effect as `MQCNO_RECONNECT_Q_MGR`.

DISABLED
Reconnection is disabled, even if requested by the client program using the MQCONNX MQI call.

Table 64. Automatic reconnection depends on the values set in the application and in the channel definition

DEFRECON	Reconnection options set in the application			
	MQCNO_RECONNECT	MQCNO_RECONNECT_Q_MGR	MQCNO_RECONNECT_AS_DEF	MQCNO_RECONNECT_DISABLED
NO	YES	QMGR	NO	NO
YES	YES	QMGR	YES	NO
QMGR	YES	QMGR	QMGR	NO
DISABLED	NO	NO	NO	NO

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the channel when an operator issues the DISPLAY CHANNEL command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

DISCINT(*integer***)**
The minimum time in seconds for which the channel waits for a message to arrive on the transmission queue, after a batch ends, before terminating the channel. A value of zero causes the message channel agent to wait indefinitely.

The value must be in the range zero through 999 999.

This parameter is valid only for channels with a channel type (CHLTYPE) of SVRCONN , SDR, SVR, CLUSSDR, CLUSRCVR.

For SVRCONN channels using the TCP protocol, this parameter is the minimum time in seconds for which the SVRCONN instance remains active without any communication from its partner client. A value of zero disables this disconnect processing. The SVRCONN inactivity interval only applies between IBM WebSphere MQ API calls from a client, so no client is disconnected during an extended MQGET with wait call. This attribute is ignored for SVRCONN channels using protocols other than TCP.

HBINT(*integer***)**

This attribute specifies the approximate time between heartbeat flows that are to be passed from a sending MCA when there are no messages on the transmission queue.

Heartbeat flows unblock the receiving MCA, which is waiting for messages to arrive or for the disconnect interval to expire. When the receiving MCA is unblocked it can disconnect the channel without waiting for the disconnect interval to expire. Heartbeat flows also free any storage buffers that have been allocated for large messages and close any queues that have been left open at the receiving end of the channel.

The value is in seconds and must be in the range 0 through 999999. A value of zero means that no heartbeat flows are to be sent. The default value is 300. To be most useful, the value needs to be less than the disconnect interval value.

For server-connection and client-connection channels, heartbeats can flow from both the server side as well as the client side independently. If no data has been transferred across the channel for the heartbeat interval, the client-connection MQI agent sends a heartbeat flow and the server-connection MQI agent responds to it with another heartbeat flow. This happens

irrespective of the state of the channel, for example, irrespective of whether it is inactive while making an API call, or is inactive waiting for client user input. The server-connection MQI agent is also capable of initiating a heartbeat to the client, again irrespective of the state of the channel. To prevent both server-connection and client-connection MQI agents heart beating to each other at the same time, the server heartbeat is flowed after no data has been transferred across the channel for the heartbeat interval plus 5 seconds.

Before IBM WebSphere MQ Version 7.0, for server-connection and client-connection channels working in the channel mode, heartbeats flow only when a server MCA is waiting for an MQGET command with the WAIT option specified, which it has issued on behalf of a client application.

For more information, see Heartbeat interval (HBINT).

KAINT(*integer***)**
The value passed to the communications stack for KeepAlive timing for this channel.

For this attribute to be effective, TCP/IP keepalive must be enabled both in the queue manager and in TCP/IP. On z/OS, you enable TCP/IP keepalive in the queue manager by issuing the ALTER QMGR TCPKEEP(YES) command; if the TCPKEEP queue manager parameter is NO, the value is ignored, and the KeepAlive facility is not used. On other platforms, TCP/IP keepalive is enabled when the KEEPALIVE=YES parameter is specified in the TCP stanza in the distributed queuing configuration file, `qm.ini`, or through the IBM WebSphere MQ Explorer.

Keepalive must also be switched on within TCP/IP itself. Refer to your TCP/IP documentation for information about configuring keepalive. On AIX, use the **no** command. On HP-UX, use the **ndd** command. On Windows, edit the registry. On z/OS, update your TCP/IP PROFILE data set and add or change the INTERVAL parameter in the TCPCONFIG section.

Although this parameter is available on all platforms, its setting is implemented only on z/OS. On platforms other than z/OS, you can access and modify the parameter, but it is only stored and forwarded; there is no functional implementation of the parameter. This functionality is useful in a clustered environment where a value set in a cluster-receiver channel definition on Solaris, for example, flows to (and is implemented by) z/OS queue managers that are in, or join, the cluster.

On platforms other than z/OS, if you need the functionality provided by the KAINT parameter, use the Heartbeat Interval (HBINT) parameter, as described in HBINT.

(*integer***)**
The KeepAlive interval to be used, in seconds, in the range 1 through 99 999.

0 The value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

AUTO
The KeepAlive interval is calculated based upon the negotiated heartbeat value as follows:

- If the negotiated HBINT is greater than zero, KeepAlive interval is set to that value plus 60 seconds.
- If the negotiated HBINT is zero, the value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

This parameter is valid for all channel types. It is ignored for channels with a TRPTYPE other than TCP or SPX.

LIKE(*channel-name***)**
The name of a channel. The parameters of this channel are used to model this definition.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from one of the following default channels, depending upon the channel type:

SYSTEM.DEF.SENDER
Sender channel

SYSTEM.DEF.SERVER
Server channel

SYSTEM.DEF.RECEIVER
Receiver channel

SYSTEM.DEF.REQUESTER
Requester channel

SYSTEM.DEF.SVRCONN
Server-connection channel

SYSTEM.DEF.CLNTCONN
Client-connection channel

SYSTEM.DEF.CLUSSDR
Cluster-sender channel

SYSTEM.DEF.CLUSRCVR
Cluster-receiver channel

This parameter is equivalent to defining the following object for a sender channel, and similarly for other channel types:

```
LIKE(SYSTEM.DEF.SENDER)
```

These default channel definitions can be altered by the installation to the default values required.

On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object and channel type you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. # LIKE is ignored if QSGDISP(COPY) is specified. However, the group object defined is used as a LIKE object.

LOCLADDR(*string*)
LOCLADDR is the local communications address for the channel. Use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. LOCLADDR might be useful in recovery scenarios where a channel is restarted on a different TCP/IP stack. LOCLADDR is also useful to force a channel to use an IPv4 or IPv6 stack on a dual-stack system. You can also use LOCLADDR to force a channel to use a dual-mode stack on a single-stack system.

This parameter is valid only for channels with a transport type (TRPTYPE) of `TCP`. If TRPTYPE is not `TCP`, the data is ignored and no error message is issued.

The value is the optional IP address, and optional port or port range used for outbound TCP/IP communications. The format for this information is as follows:

Table 73 on page 452 shows how the LOCLADDR parameter can be used:
LOCLADDR([ip-addr][(low-port[,high-port])]*[,[ip-addr][(low-port[,high-port])]]*)

The maximum length of LOCLADDR, including multiple addresses, is MQ_LOCAL_ADDRESS_LENGTH.

If you omit LOCLADDR, a local address is automatically allocated.

Note, that you can set LOCLADDR for a C client using the Client Channel Definition Table (CCDT).

All the parameters are optional. Omitting the ip-addr part of the address is useful to enable the configuration of a fixed port number for an IP firewall. Omitting the port number is useful to select a particular network adapter without having the identify a unique local port number. The TCP/IP stack generates a unique port number.

Specify *[,[ip-addr][(low-port[,high-port])]]* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *[,[ip-addr][(low-port[,high-port])]]* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

ip-addr
ip-addr is specified in one of three forms:

IPv4 dotted decimal
For example 192.0.2.1

IPv6 hexadecimal notation
For example 2001:DB8:0:0:0:0:0:0

Alphanumeric host name form
For example WWW.EXAMPLE.COM

low-port and high-port
low-port and high-port are port numbers enclosed in parentheses.

Table 65. Examples of how the LOCLADDR parameter can be used

LOCLADDR	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98, 9.20.4.99	Channel binds to either IP address. The address might be two network adapters on one server, or a different network adapter on two different servers in a multi-instance configuration.
9.20.4.98(1000)	Channel binds to this address and port 1000 locally
9.20.4.98(1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to port in range 1000 - 2000 locally

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, CLUSRCVR, or MQTT.

On CLUSSDR channels, the IP address and port to which the outbound channel binds, is a combination of fields. It is a concatenation of the IP address, as defined in the LOCLADDR parameter, and the port range from the cluster cache. If there is no port range in the cache, the port range defined in the LOCLADDR parameter is used. This port range does not apply to z/OS.

Even though this parameter is similar in form to CONNAME, it must not be confused with it. The LOCLADDR parameter specifies the characteristics of the local communications, whereas the CONNAME parameter specifies how to reach a remote queue manager.

When a channel is started, the values specified for CONNAME and LOCLADDR determine the IP stack to be used for communication; see Table 3 and Local Address (LOCLADDR) .

If the TCP/IP stack for the local address is not installed or configured, the channel does not start and an exception message is generated. The message indicates that the connect() request specifies an interface address that is not known on the default IP stack. To direct the connect() request to the alternative stack, specify the **LOCLADDR** parameter in the channel definition as either an interface on the alternative stack, or a DNS host name. The same specification also works for listeners that might not use the default stack. To find the value to code for **LOCLADDR**, run the **NETSTAT HOME** command on the IP stacks that you want to use as alternatives.

For channels with a channel type (CHLTYPE) of MQTT the usage of this parameter is slightly different. Specifically, a telemetry channel (MQTT) **LOCLADDR** parameter expects only an IPv4 or IPv6 IP address, or a valid host name as a string. This string must not contain a port number or port range. If an IP address is entered, only the address format is validated. The IP address itself is not validated.

Table 66. How the IP stack to be used for communication is determined

Protocols supported	CONNAME	LOCLADDR	Action of channel
IPv4 only	IPv4 address[1]		Channel binds to IPv4 stack
	IPv6 address[2]		Channel fails to resolve CONNAME
	IPv4 and 6 host name[3]		Channel binds to IPv4 stack
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	Any address[4]	IPv6 address	Channel fails to resolve LOCLADDR
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv4 stack
IPv4 and IPv6	IPv4 address		Channel binds to IPv4 stack
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to stack determined by IPADDRV
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	IPv4 address	IPv6 address	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv6 address	Channel binds IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to stack determined by IPADDRV

Table 66. How the IP stack to be used for communication is determined (continued)

Protocols supported	CONNAME	LOCLADDR	Action of channel
IPv6 only	IPv4 address		Channel maps CONNAME to IPv6[5]
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to IPv6 stack
	Any address	IPv4 address	Channel fails to resolve LOCLADDR
	IPv4 address	IPv6 address	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv6 address	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds to IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv6 stack

Notes:

1. IPv4 address. An IPv4 host name that resolves only to an IPv4 network address or a specific dotted notation IPv4 address, for example `1.2.3.4`. This note applies to all occurrences of 'IPv4 address' in this table.
2. IPv6 address. An IPv6 host name that resolves only to an IPv6 network address or a specific hexadecimal notation IPv6 address, for example `4321:54bc`. This note applies to all occurrences of 'IPv6 address' in this table.
3. IPv4 and 6 host name. A host name that resolves to both IPv4 and IPv6 network addresses. This note applies to all occurrences of 'IPv4 and 6 host name' in this table.
4. Any address. IPv4 address, IPv6 address, or IPv4 and 6 host name. This note applies to all occurrences of 'Any address' in this table.
5. Maps IPv4 CONNAME to IPv4 mapped IPv6 address. IPv6 stack implementations that do not support IPv4 mapped IPv6 addressing fail to resolve the CONNAME. Mapped addresses might require protocol translators in order to be used. The use of mapped addresses is not recommended.

LONGRTY(*integer***)**

When a sender, server, or cluster-sender channel is attempting to connect to the remote queue manager, and the count specified by SHORTRTY has been exhausted, this parameter specifies the maximum number of further attempts that are made to connect to the remote queue manager, at intervals specified by LONGTMR.

If this count is also exhausted without success, an error is logged to the operator, and the channel is stopped. The channel must then be restarted with a command (it is not started automatically by the channel initiator).

The value must be in the range zero through 999999999.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

LONGTMR(*integer***)**

For long retry attempts, this parameter is the maximum number of seconds to wait before reattempting connection to the remote queue manager.

The time is approximate; zero means that another connection attempt is made as soon as possible.

The interval between retries might be extended if the channel has to wait to become active.

The value must be in the range zero through 999999999.

Note: For implementation reasons, the maximum retry interval that can be used is 999,999; values exceeding this maximum are treated as 999,999. Similarly, the minimum retry interval that can be used is 2; values less than this minimum are treated as 2.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

MAXINST(*integer***)**
The maximum number of simultaneous instances of an individual server-connection channel that can be started.

The value must be in the range zero through 999999999.

A value of zero prevents all client access on this channel.

If the value of this parameter is reduced to a number that is less than the number of instances of the server-connection channel that are currently running, then those running instances are not affected. However, new instances cannot start until sufficient existing instances have ceased to run so that the number of currently running instances is less than the value of this parameter.

On z/OS, without the Client Attachment feature installed, a maximum of five instances are allowed on the channel named SYSTEM.ADMIN.SVRCONN. If MAXINST is set to a larger number than five, it is interpreted as zero without the CAF installed.

This parameter is valid only for channels with a channel type (CHLTYPE) of SVRCONN.

MAXINSTC(*integer***)**
The maximum number of simultaneous individual server-connection channels that can be started from a single client. In this context, connections that originate from the same remote network address are regarded as coming from the same client.

The value must be in the range zero through 999999999.

A value of zero prevents all client access on this channel.

If the value of this parameter is reduced to a number that is less than the number of instances of the server-connection channel that is currently running from individual clients, then those running instances are not affected. However, new instances from those clients cannot start until sufficient instances have ceased to run that the number of running instances is less than the value of this parameter.

On z/OS, without the Client Attachment feature installed, only a maximum of five instances are allowed on the channel named SYSTEM.ADMIN.SVRCONN.

This parameter is valid only for channels with a channel type (CHLTYPE) of SVRCONN.

MAXMSGL(*integer***)**
Specifies the maximum message length that can be transmitted on the channel. This parameter is compared with the value for the partner and the actual maximum used is the lower of the two values. The value is ineffective if the MQCB function is being executed and the channel type (CHLTYPE) is SVRCONN.

The value zero means the maximum message length for the queue manager.

On platforms other than z/OS, specify a value in the range zero through to the maximum message length for the queue manager.

On z/OS, specify a value in the range zero through 104857600 bytes (100 MB).

See the MAXMSGL parameter of the ALTER QMGR command for more information.

MCANAME(*string***)**
Message channel agent name.

This parameter is reserved, and if specified must only be set to blanks (maximum length 20 characters).

MCATYPE
Specifies whether the message-channel-agent program on an outbound message channel runs as a thread or a process.

PROCESS
The message channel agent runs as a separate process.

THREAD The message channel agent runs as a separate thread

In situations where a threaded listener is required to service many incoming requests, resources can become strained. In this case, use multiple listener processes and target incoming requests at specific listeners though the port number specified on the listener.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLUSSDR, or CLUSRCVR. It is not supported on z/OS.

On z/OS, it is supported only for channels with a channel type of CLUSRCVR. When specified in a CLUSRCVR definition, MCATYPE is used by a remote machine to determine the corresponding CLUSSDR definition.

MCAUSER(*string***)**
Message channel agent user identifier.

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL). For more details, see Channel authentication records.

This parameter interacts with PUTAUT, see the definition of that parameter for more information.

If it is nonblank, it is the user identifier that is to be used by the message channel agent for authorization to access IBM WebSphere MQ resources, including (if PUTAUT is DEF) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

The default user identifier is derived from the user ID that started the receiving channel. The possible values are:

- On z/OS, the user ID assigned to the channel-initiator started task by the z/OS started-procedures table.
- For TCP/IP, other than z/OS, the user ID from the `inetd.conf` entry, or the user that started the listener.
- For SNA, other than z/OS, the user ID from the SNA server entry or, in the absence of this user ID the incoming attach request, or the user that started the listener.
- For NetBIOS or SPX, the user ID that started the listener.

The maximum length of the string is 64 characters on Windows and 12 characters on other platforms. On Windows, you can optionally qualify a user identifier with the domain name in the format `user@domain`.

This parameter is not valid for channels with a channel type (CHLTYPE) of SDR, SVR, CLNTCONN, CLUSSDR.

MODENAME(*string***)**
LU 6.2 mode name (maximum length 8 characters).

This parameter is valid only for channels with a transport type (TRPTYPE) of LU 6.2. If TRPTYPE is not LU 6.2, the data is ignored and no error message is issued.

If specified, this parameter must be set to the SNA mode name unless the CONNAME contains a side-object name, in which case it must be set to blanks. The actual name is then taken from the CPI-C Communications Side Object, or APPC side information data set.

This parameter is not valid for channels with a channel type (CHLTYPE) of RCVR or SVRCONN.

MONCHL

Controls the collection of online monitoring data for channels:

QMGR Collect monitoring data according to the setting of the queue manager parameter MONCHL.

OFF Monitoring data collection is turned off for this channel.

LOW If the value of the queue manager MONCHL parameter is not NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MEDIUM If the value of the queue manager MONCHL parameter is not NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.

HIGH If the value of the queue manager MONCHL parameter is not NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

Changes to this parameter take effect only on channels started after the change occurs.

For cluster channels, the value of this parameter is not replicated in the repository and, therefore, not used in the auto-definition of cluster-sender channels. For auto-defined cluster-sender channels, the value of this parameter is taken from the queue manager attribute MONACLS. This value might then be overridden in the channel auto-definition exit.

MRDATA(*string***)**

Channel message-retry exit user data. The maximum length is 32 characters.

This parameter is passed to the channel message-retry exit when it is called.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, or CLUSRCVR.

MREXIT(*string***)**

Channel message-retry exit name.

The format and maximum length of the name is the same as for MSGEXIT, however you can only specify one message-retry exit.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, or CLUSRCVR.

MRRTY(*integer***)**

The number of times the channel tries again before it decides it cannot deliver the message.

This parameter controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRRTY is passed to the exit to use, but the number of retries performed (if any) is controlled by the exit, and not by this parameter.

The value must be in the range zero through 999999999. A value of zero means that no retries are performed.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, or CLUSRCVR.

MRTMR(*integer***)**

The minimum interval of time that must pass before the channel can try the MQPUT operation again. This time interval is in milliseconds.

This parameter controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRTMR is passed to the exit to use, but the retry interval is controlled by the exit, and not by this parameter.

The value must be in the range zero through 999 999 999. A value of zero means that the retry is performed as soon as possible (if the value of MRRTY is greater than zero).

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, or CLUSRCVR.

MSGDATA(*string***)**
User data for the channel message exit. The maximum length is 32 characters.

This data is passed to the channel message exit when it is called.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first message exit specified, the second string to the second exit, and so on.

On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first message exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of message exit data for each channel.

Note: This parameter is accepted but ignored for server-connection and client-connection channels.

MSGEXIT(*string***)**
Channel message exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after a message has been retrieved from the transmission queue (sender or server), or immediately before a message is put to a destination queue (receiver or requester).

  The exit is given the entire application message and transmission queue header for modification.
- At initialization and termination of the channel.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one message exit name for each channel.

For channels with a channel type (CHLTYPE) of CLNTCONN or SVRCONN, this parameter is accepted but ignored, because message exits are not invoked for such channels.

The format and maximum length of the name depends on the environment:

- On UNIX and Linux systems, it is of the form:

  ```
  libraryname(functionname)
  ```

  The maximum length of the string is 128 characters.
- On Windows, it is of the form:

  ```
  dllname(functionname)
  ```

  where *dllname* is specified without the suffix (".DLL"). The maximum length of the string is 128 characters.
- On IBM i, it is of the form:

  ```
  progname libname
  ```

where *program name* occupies the first 10 characters and *libname* the second 10 characters (both padded to the right with blanks if necessary). The maximum length of the string is 20 characters.

- On z/OS, it is a load module name, maximum length 8 characters (128 characters are allowed for exit names for client-connection channels, subject to a maximum total length including commas of 999).

NETPRTY(*integer***)**
The priority for the network connection. Distributed queuing chooses the path with the highest priority if there are multiple paths available. The value must be in the range zero through 9; zero is the lowest priority.

This parameter is valid only for CLUSRCVR channels.

NPMSPEED
The class of service for nonpersistent messages on this channel:

FAST Fast delivery for nonpersistent messages; messages might be lost if the channel is lost. Messages are retrieved using MQGMO_SYNCPOINT_IF_PERSISTENT and so are not included in the batch unit of work.

NORMAL Normal delivery for nonpersistent messages.

If the sending side and the receiving side do not agree about this parameter, or one does not support it, NORMAL is used.

This parameter is valid only for channels with a CHLTYPE of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

PASSWORD(*string***)**
Password used by the message channel agent when attempting to initiate a secure LU 6.2 session with a remote message channel agent. The maximum length is 12 characters.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLNTCONN, or CLUSSDR. On z/OS, it is supported only for channels with a channel type (CHLTYPE) of CLNTCONN.

Although the maximum length of the parameter is 12 characters, only the first 10 characters are used.

PROPCTL
Property control attribute.

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

This parameter is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

This parameter is optional.

Permitted values are:

COMPAT
COMPAT allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

Message properties	Result
The message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All optional message properties (where the **Support** value is MQPD_SUPPORT_OPTIONAL), except properties in the message descriptor or extension, are placed in one or more MQRFH2 headers in the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All message properties, except properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL	The message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.
The message contains one or more properties where the **Support** field of the property descriptor is set to MQPD_SUPPORT_OPTIONAL but other fields of the property descriptor are set to non-default values.	The properties with non-default values are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the *content='properties'* attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a V6 or prior queue manager.

NONE
All properties of the message, except properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

If the message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL then the message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.

ALL All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

PUTAUT
Specifies which user identifiers are used to establish authority to put messages to the destination queue (for messages channels) or to execute an MQI call (for MQI channels).

DEF The default user ID is used. On z/OS, DEF might involve using both the user ID received from the network and that derived from MCAUSER.

CTX The user ID from the *UserIdentifier* field of the message descriptor is used. On z/OS, CTX might involve also using the user ID received from the network or that derived from MCAUSER, or both.

ONLYMCA
The default user ID is used. Any user ID received from the network is not used. This value is supported only on z/OS.

ALTMCA The user ID from the *UserIdentifier* field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS.

On z/OS, the user IDs that are checked, and how many user IDs are checked, depends on the setting of the MQADMIN RACF® class hlq.RESLEVEL profile. Depending on the level of access the user ID of the channel initiator has to hlq.RESLEVEL, zero, one or two user IDs are checked.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, CLUSRCVR, or, on z/OS only, SVRCONN. CTX and ALTMCA are not valid for SVRCONN channels.

QMNAME(*string***)**
Queue manager name.

For channels with a channel type (CHLTYPE) of CLNTCONN, this parameter is the name of a queue manager to which an application that is running in a client environment and using the client channel definition table can request connection. This parameter need not be the name of the queue manager on which the channel is defined, to allow a client to connect to different queue managers.

For channels of other types, this parameter is not valid.

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero:
	`DEFINE CHANNEL(channel-name) CHLTYPE(type)`
	`REPLACE QSGDISP(COPY)`
	
	The ALTER for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with QSGDISP(QMGR) or QSGDISP(COPY). Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

RCVDATA(*string***)**

Channel receive exit user data (maximum length 32 characters).

This parameter is passed to the channel receive exit when it is called.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first receive exit specified, the second string to the second exit, and so on.

On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first receive exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of receive exit data for each channel.

RCVEXIT(*string***)**

Channel receive exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before the received network data is processed.

The exit is given the complete transmission buffer as received. The contents of the buffer can be modified as required.

- At initialization and termination of the channel.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one receive exit name for each channel.

The format and maximum length of the name is the same as for MSGEXIT.

REPLACE and NOREPLACE
Whether the existing definition (and on z/OS, with the same disposition) is to be replaced with this one. This parameter is optional. Any object with a different disposition is not changed.

REPLACE
The definition replaces any existing definition of the same name. If a definition does not exist, one is created. REPLACE does not alter the channel status.

NOREPLACE
The definition does not replace any existing definition of the same name.

SCYDATA(*string***)**
Channel security exit user data (maximum length 32 characters).

This parameter is passed to the channel security exit when it is called.

SCYEXIT(*string***)**
Channel security exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is able to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote queue manager are given to the exit.
- At initialization and termination of the channel.

The format and maximum length of the name is the same as for MSGEXIT but only one name is allowed.

SENDDATA(*string***)**
Channel send exit user data. The maximum length is 32 characters.

This parameter is passed to the channel send exit when it is called.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first send exit specified, the second string to the second exit, and so on.

On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first send exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of send exit data for each channel.

SENDEXIT(*string***)**
Channel send exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before data is sent out on the network.

  The exit is given the complete transmission buffer before it is transmitted. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one send exit name for each channel.

The format and maximum length of the name is the same as for MSGEXIT.

SEQWRAP(*integer***)**
When this value is reached, sequence numbers wrap to start again at 1.

This value is nonnegotiable and must match in both the local and remote channel definitions.

The value must be in the range 100 through 999999999.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

SHARECNV(*integer***)**
Specifies the maximum number of conversations that can be sharing each TCP/IP channel instance. A SHARECNV value of:

1 Specifies no sharing of conversations over a TCP/IP channel instance. Client heartbeating is available whether in an MQGET call or not. Read ahead and client asynchronous consumption are also available, and channel quiescing is more controllable.

0 Specifies no sharing of conversations over a TCP/IP channel instance. The channel instance runs in a mode before that of IBM WebSphere MQ Version 7.0, regarding:
- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Client asynchronous consumption

The value must be in the range zero through 999999999.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLNTCONN or SVRCONN. If the client-connection SHARECNV value does not match the server-connection SHARECNV value, the lower of the two values is used. This parameter is ignored for channels with a transport type (TRPTYPE) other than TCP.

All the conversations on a socket are received by the same thread.

High SHARECNV limits have the advantage of reducing queue manager thread usage. However, if many conversations sharing a socket are all busy, there is a possibility of delays as the conversations contend with one another to use the receiving thread. In this situation, a lower SHARECNV value is better.

The number of shared conversations does not contribute to the MAXINST or MAXINSTC totals.

Note: You should restart the client for this change to take effect.

SHORTRTY(*integer***)**
The maximum number of attempts that are made by a sender, server, or cluster-sender channel to connect to the remote queue manager, at intervals specified by SHORTTMR, before the (normally longer) LONGRTY and LONGTMR are used.

Retry attempts are made if the channel fails to connect initially (whether it is started automatically by the channel initiator or by an explicit command), and also if the connection fails after the channel has successfully connected. However, if the cause of the failure is such that more attempts are unlikely to be successful, they are not attempted.

The value must be in the range zero through 999999999.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

SHORTTMR(*integer***)**
For short retry attempts, this parameter is the maximum number of seconds to wait before reattempting connection to the remote queue manager.

The time is approximate; zero means that another connection attempt is made as soon as possible.

The interval between retries might be extended if the channel has to wait to become active.

The value must be in the range zero through 999999999.

Note: For implementation reasons, the maximum retry interval that can be used is 999999; values exceeding this maximum are treated as 999999. Similarly, the minimum retry interval that can be used is 2; values less than this minimum are treated as 2.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

SSLCAUTH
Defines whether IBM WebSphere MQ requires a certificate from the SSL client. The initiating end of the channel acts as the SSL client, so this parameter applies to the end of the channel that receives the initiation flow, which acts as the SSL server.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, SVRCONN, CLUSRCVR, SVR, or RQSTR.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

REQUIRED
IBM WebSphere MQ requires and validates a certificate from the SSL client.

OPTIONAL
The peer SSL client system might still send a certificate. If it does, the contents of this certificate are validated as normal.

SSLCIPH*(string)*
SSLCIPH specifies the CipherSpec that is used on the channel. The maximum length is 32 characters. This parameter is valid on all channel types which use transport type `TRPTYPE(TCP)`. If the SSLCIPH parameter is blank, no attempt is made to use SSL on the channel.

Note: When SSLCIPH is used with a telemetry channel, it means “SSL Cipher Suite”. See the SSLCIPH description in “ALTER CHANNEL (MQTT)”.

Specify the name of the CipherSpec you are using. The CipherSpecs that can be used with WebSphere MQ SSL support are shown in the following table. The SSLCIPH values must specify

the same CipherSpec on both ends of the channel.

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
NULL_MD5 [a]	SSL 3.0	MD5	None	0	No	No	No
NULL_SHA [a]	SSL 3.0	SHA-1	None	0	No	No	No
RC4_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC4	40	No	No	No
RC4_MD5_US [a]	SSL 3.0	MD5	RC4	128	No	No	No
RC4_SHA_US [a]	SSL 3.0	SHA-1	RC4	128	No	No	No
RC2_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC2	40	No	No	No
DES_SHA_EXPORT [2] [a]	SSL 3.0	SHA-1	DES	56	No	No	No
RC4_56_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	RC4	56	No	No	No
DES_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	DES	56	No	No	No
TLS_RSA_WITH_AES_128_CBC_SHA [a]	TLS 1.0	SHA-1	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA [4] [a]	TLS 1.0	SHA-1	AES	256	Yes	No	No
TLS_RSA_WITH_DES_CBC_SHA [a]	TLS 1.0	SHA-1	DES	56	No[5]	No	No
FIPS_WITH_DES_CBC_SHA [b]	SSL 3.0	SHA-1	DES	56	No[6]	No	No
TLS_RSA_WITH_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	256	Yes	No	No
ECDHE_ECDSA_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No
ECDHE_RSA_RC4_128_SHA256 [b]	TLS 1.2	SHA_1	RC4	128	No	No	No
ECDHE_ECDSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_ECDSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_RSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_RSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_ECDSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	Yes	No
ECDHE_ECDSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	Yes
ECDHE_RSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
ECDHE_RSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_NULL_SHA256 [b]	TLS 1.2	SHA-256	None	0	No	No	No
ECDHE_RSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
ECDHE_ECDSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No
TLS_RSA_WITH_NULL_NULL [b]	TLS 1.2	None	None	0	No	No	No
TLS_RSA_WITH_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No

Notes:

1. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
2. The maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
3. The handshake key size is 1024 bits.
4. This CipherSpec cannot be used to secure a connection from the WebSphere MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
5. This CipherSpec was FIPS 140-2 certified before 19 May 2007.
6. This CipherSpec was FIPS 140-2 certified before 19 May 2007. The name FIPS_WITH_DES_CBC_SHA is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. This CipherSpec is deprecated and its use is not recommended.
7. This CipherSpec can be used to transfer up to 32 GB of data before the connection is terminated with error AMQ9288. To avoid this error, either avoid using triple DES, or enable secret key reset when using this CipherSpec.

Platform support:

- a Available on all supported platforms.
- b Available only on UNIX, Linux, and Windows platforms.

When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the SSL handshake can depend on the size stored in the certificate and on the CipherSpec:

- On z/OS, Windows, UNIX and Linux systems, when a CipherSpec name includes _EXPORT, the maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
- On Windows, UNIX and Linux systems, when a CipherSpec name includes _EXPORT1024, the handshake key size is 1024 bits.
- Otherwise the handshake key size is the size stored in the certificate.

SSLPEER(*string***)**

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the SSL certificate.) If the Distinguished Name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

Note: An alternative way of restricting connections into channels by matching against the SSL or TLS Subject Distinguished Name, is to use channel authentication records. With channel authentication records, different SSL or TLS Subject Distinguished Name patterns can be applied to the same channel. If both SSLPEER on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns in order to connect. For more information, see Channel authentication records.

This parameter is optional; if it is not specified, the Distinguished Name of the peer is not checked at channel start up. (The Distinguished Name from the certificate is still written into the SSLPEER definition held in memory, and passed to the security exit). If SSLCIPH is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The SSLPEER value is specified in the standard form used to specify a Distinguished Name. For example:

```
SSLPEER('SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN="H1_C_FR1",O=IBM,C=GB')
```

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

Table 67. Attribute types supported by SSLPEER.

A two column table descibing the attributes supported by the SSLPEER attribute

Summary attribute	**Description**
SERIALNUMBER	Certificate serial number
MAIL	Email address
E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zip code
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

IBM WebSphere MQ accepts only uppercase letters for the attribute types.

If any of the unsupported attribute types are specified in the SSLPEER string, an error is output either when the attribute is defined or at run time (depending on which platform you are running on), and the string is deemed not to have matched the Distinguished Name of the flowed certificate.

If the Distinguished Name of the flowed certificate contains multiple OU (organizational unit) attributes, and SSLPEER specifies these attributes to be compared, they must be defined in descending hierarchical order. For example, if the Distinguished Name of the flowed certificate contains the OUs `OU=Large Unit, OU=Medium Unit, OU=Small Unit`, specifying the following SSLPEER values works:

```
('OU=Large Unit,OU=Medium Unit')
('OU=*,OU=Medium Unit,OU=Small Unit')
('OU=*,OU=Medium Unit')
```

but specifying the following SSLPEER values fails:

```
('OU=Medium Unit,OU=Small Unit')
('OU=Large Unit,OU=Small Unit')
('OU=Medium Unit')
('OU=Small Unit, Medium Unit, Large Unit')
```

As indicated in these examples, attributes at the low end of the hierarchy can be omitted. For example, `('OU=Large Unit,OU=Medium Unit')` is equivalent to `('OU=Large Unit,OU=Medium Unit,OU=*')`

If two DNs are equal in all respects except for their DC values, the same matching rules apply as for OUs except that in DC values the left-most DC is the lowest-level (most specific) and the comparison ordering differs accordingly.

Any or all the attribute values can be generic, either an asterisk (*) on its own, or a stem with initiating or trailing asterisks. Asterisks allow the SSLPEER to match any Distinguished Name value, or any value starting with the stem for that attribute.

If an asterisk is specified at the beginning or end of any attribute value in the Distinguished Name on the certificate, you can specify '*' to check for an exact match in SSLPEER. For example, if you have an attribute of `CN='Test*'` in the Distinguished Name of the certificate, you can use the following command:

```
SSLPEER('CN=Test\*')
```

The maximum length of the parameter is 1024 bytes on Windows, IBM i, UNIX and Linux platforms, and 256 bytes on z/OS.

STATCHL

Controls the collection of statistics data for channels:

QMGR The value of the STATCHL parameter of the queue manager is inherited by the channel.

OFF Statistics data collection is turned off for this channel.

LOW If the value of the STATCHL parameter of the queue manager is not NONE, statistics data collection is turned on, with a low rate of data collection, for this channel.

MEDIUM If the value of the STATCHL parameter of the queue manager is not NONE, statistics data collection is turned on, with a moderate rate of data collection, for this channel.

HIGH If the value of the STATCHL parameter of the queue manager is not NONE, statistics data collection is turned on, with a high rate of data collection, for this channel.

Changes to this parameter take effect only on channels started after the change occurs.

For cluster channels, the value of this parameter is not replicated in the repository and used in the auto-definition of cluster-sender channels. For auto-defined cluster-sender channels, the value of this parameter is taken from the attribute STATACLS of the queue manager. This value might then be overridden in the channel auto-definition exit.

This parameter is valid only on AIX, IBM i, HP-UX, Linux, Solaris, and Windows.

TPNAME(*string***)**

LU 6.2 transaction program name (maximum length 64 characters).

This parameter is valid only for channels with a transport type (TRPTYPE) of LU 6.2.

Set this parameter to the SNA transaction program name, unless the CONNAME contains a side-object name in which case set it to blanks. The actual name is taken instead from the CPI-C Communications Side Object, or the APPC side information data set.

On Windows SNA Server, and in the side object on z/OS, the TPNAME is wrapped to uppercase.

This parameter is not valid for channels with a channel type (CHLTYPE) of RCVR.

TRPTYPE
Transport type to be used.

On AIX, IBM i, HP-UX, Linux, Solaris, and Windows, and z/OS, this parameter is optional because, if you do not enter a value, the value specified in the SYSTEM.DEF.*channel-type* definition is used. However, no check is made that the correct transport type has been specified if the channel is initiated from the other end. On z/OS, if the SYSTEM.DEF.*channel-type* definition does not exist, the default is LU62.

This parameter is required on all other platforms.

LU62 SNA LU 6.2

NETBIOS
NetBIOS (supported only on Windows, and DOS; it also applies to z/OS for defining client-connection channels that connect to servers on the platforms supporting NetBIOS)

SPX Sequenced packet exchange (supported only on Windows, and DOS; it also applies to z/OS for defining client-connection channels that connect to servers on the platforms supporting SPX)

TCP Transmission Control Protocol - part of the TCP/IP protocol suite

USEDLQ
Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

NO Messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the NPMSPEED setting.

YES When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used, else the behavior is as for NO. YES is the default value.

USERID(*string***)**
Task user identifier. The maximum length is 12 characters.

This parameter is used by the message channel agent when attempting to initiate a secure LU 6.2 session with a remote message channel agent.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLNTCONN, or CLUSSDR. On z/OS, it is supported only for CLNTCONN channels.

Although the maximum length of the parameter is 12 characters, only the first 10 characters are used.

On the receiving end, if passwords are kept in encrypted format and the LU 6.2 software is using a different encryption method, an attempt to start the channel fails with invalid security details. You can avoid invalid security details by modifying the receiving SNA configuration to either:

- Turn off password substitution, or
- Define a security user ID and password.

XMITQ(*string***)**
Transmission queue name.

The name of the queue from which messages are retrieved. See Rules for naming IBM WebSphere MQ objects.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR or SVR. For these channel types, this parameter is required.

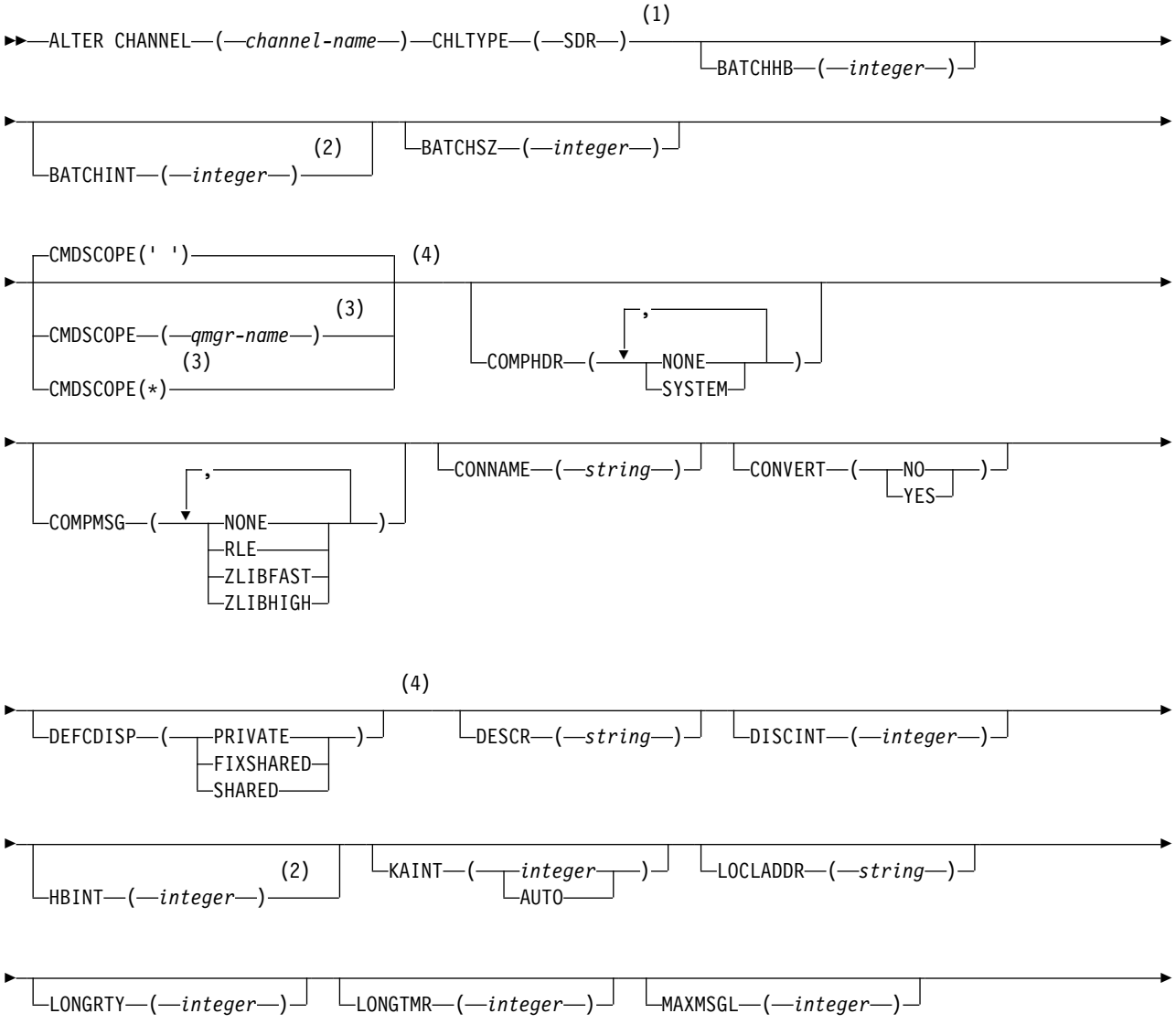
There is a separate syntax diagram for each type of channel:

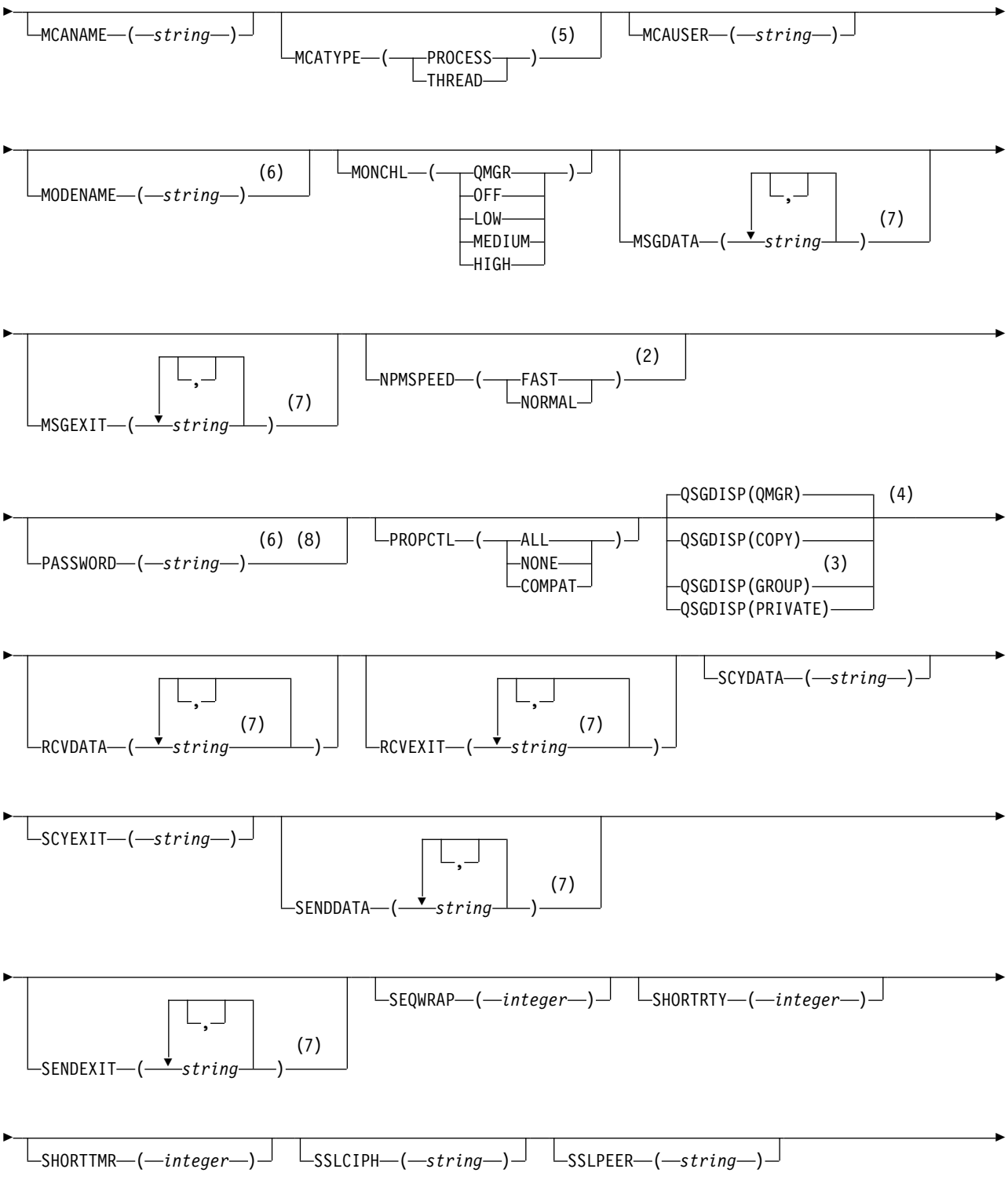
- "Sender channel"
- "Server channel" on page 319
- "Receiver channel" on page 322
- "Requester channel" on page 324
- "Client-connection channel" on page 327
- "Server-connection channel" on page 328
- "Cluster-sender channel" on page 330
- "Cluster-receiver channel" on page 332

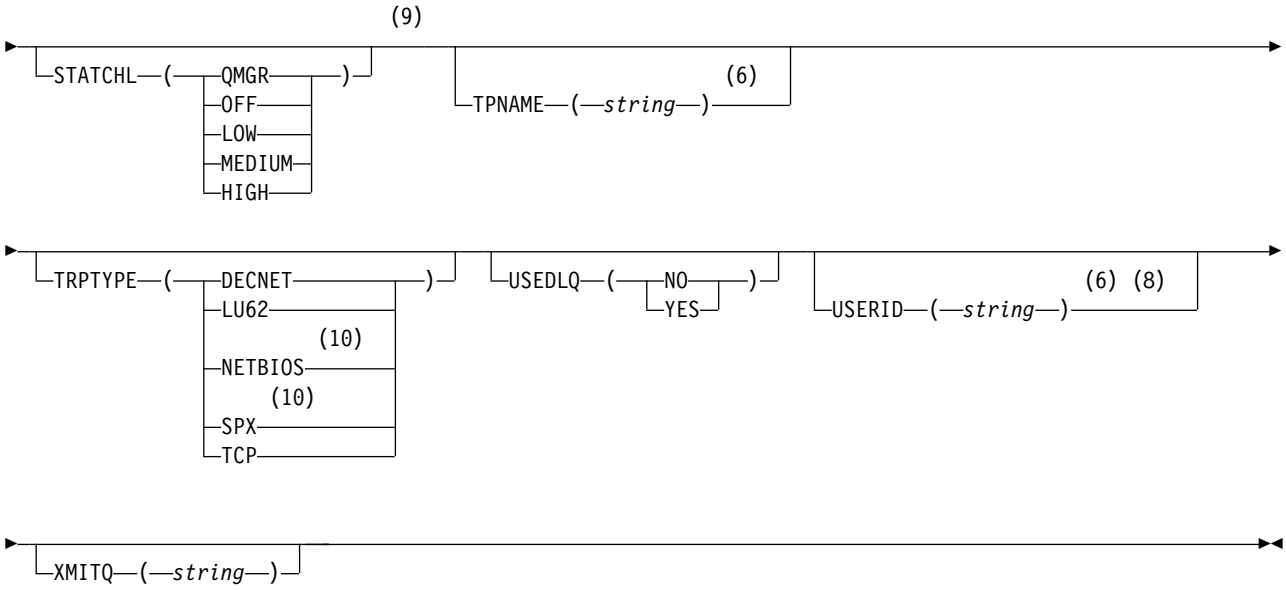
Sender channel:

Syntax diagram for a sender channel when using the ALTER CHANNEL command.

ALTER CHANNEL




MCANAME—(—string—)
MCATYPE—(—PROCESS—)—(5)
THREAD
MCAUSER—(—string—)
MODENAME—(—string—)—(6)
MONCHL—(—QMGR—)
OFF
LOW
MEDIUM
HIGH
MSGDATA—(—string—)—(7)
MSGEXIT—(—string—)—(7)
NPMSPEED—(—FAST—)—(2)
NORMAL
PASSWORD—(—string—)—(6) (8)
PROPCTL—(—ALL—)
NONE
COMPAT
QSGDISP(QMGR)—(4)
QSGDISP(COPY)
QSGDISP(GROUP)—(3)
QSGDISP(PRIVATE)
RCVDATA—(—string—)—(7)
RCVEXIT—(—string—)—(7)
SCYDATA—(—string—)
SCYEXIT—(—string—)
SENDDATA—(—string—)—(7)
SENDEXIT—(—string—)—(7)
SEQWRAP—(—integer—)
SHORTRTY—(—integer—)
SHORTTMR—(—integer—)
SSLCIPH—(—string—)
SSLPEER—(—string—)



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows and z/OS.

3 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

4 Valid only on z/OS.

5 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

6 Valid only if TRPTYPE is LU62.

7 You can specify more than one value on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS only.

8 Not valid on z/OS.

9 This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

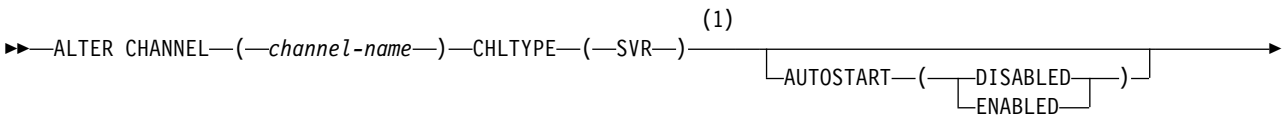
10 Valid only Windows.

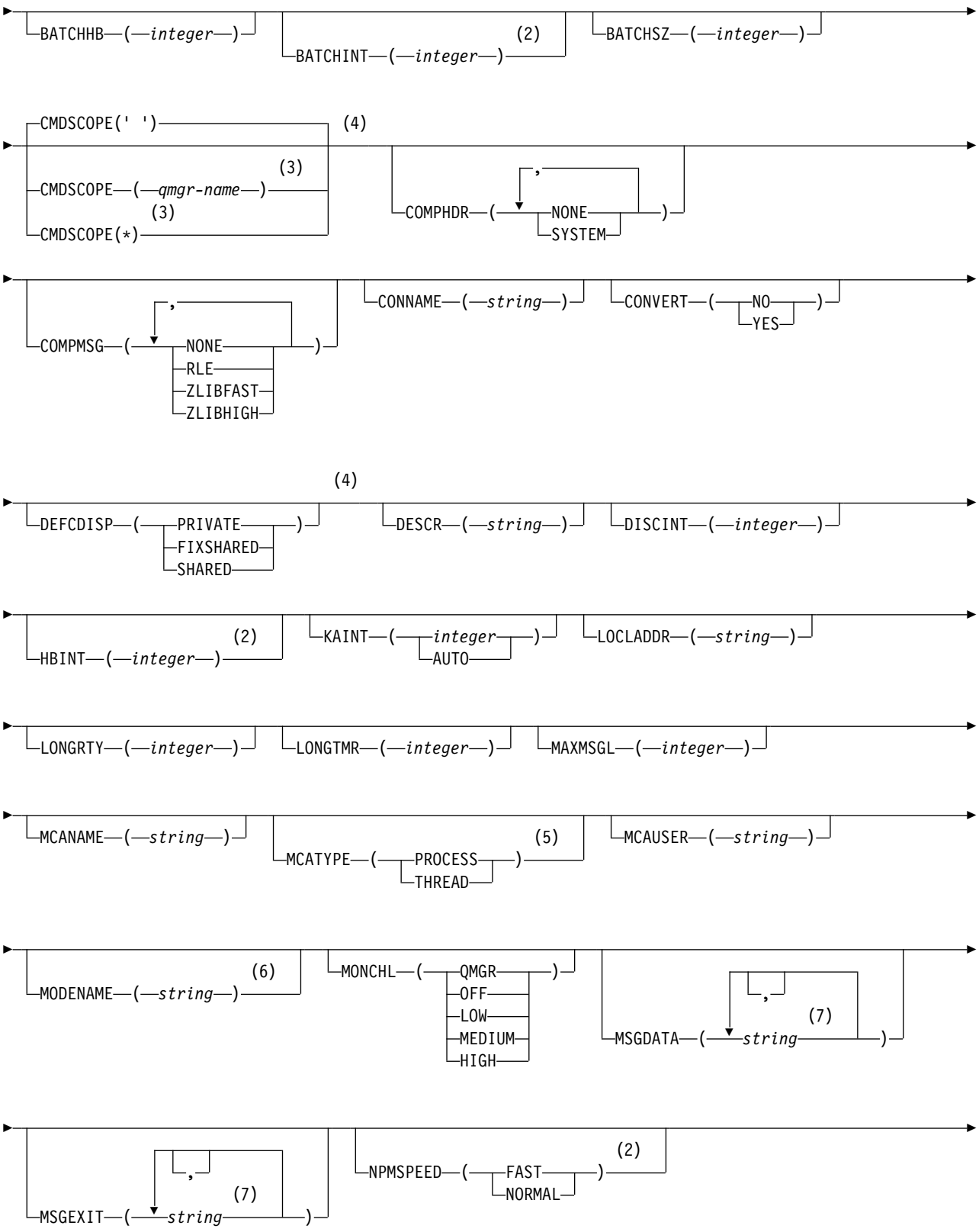
The parameters are described in "ALTER CHANNEL" on page 285.

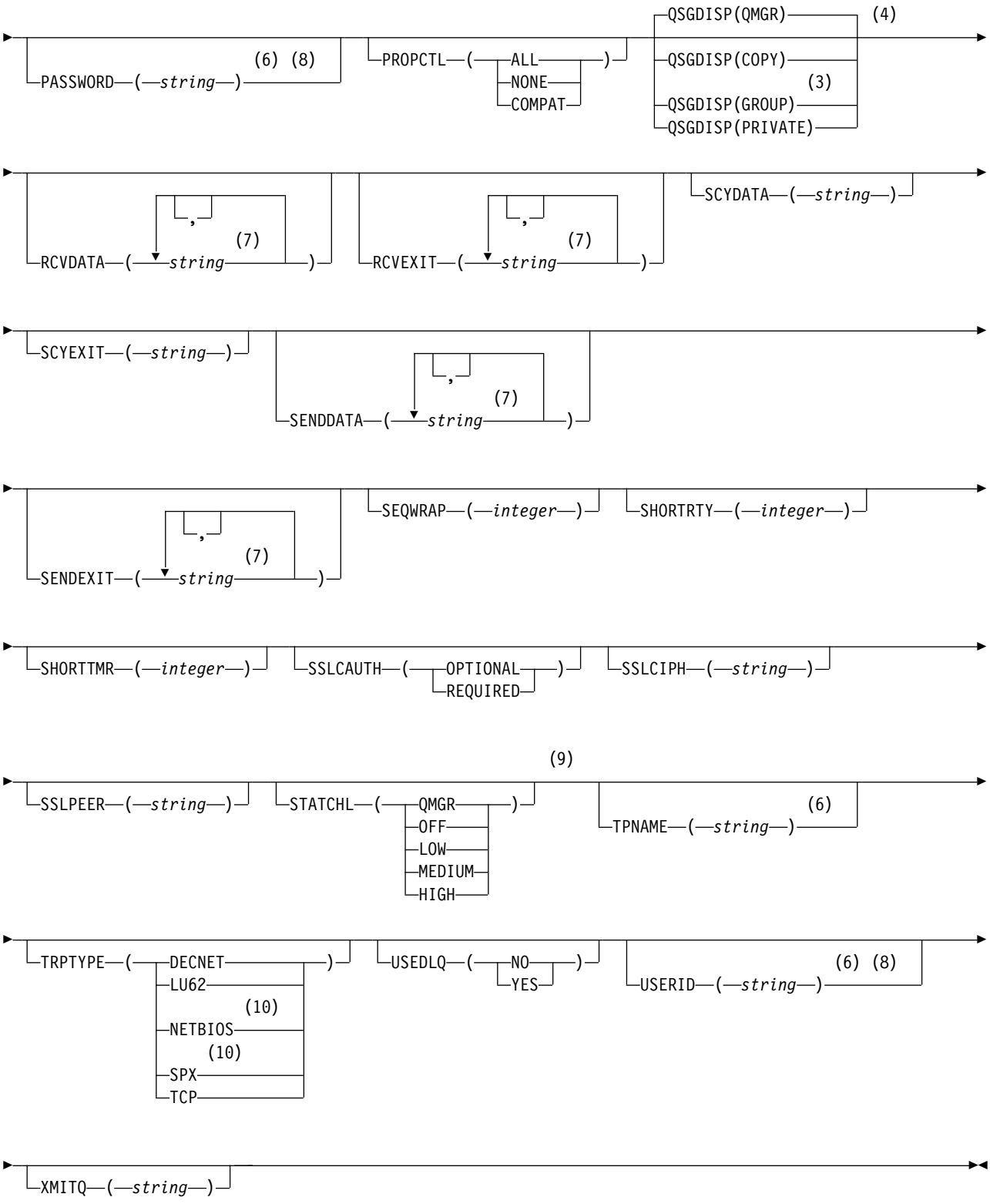
Server channel:

Syntax diagram for a server channel when using the ALTER CHANNEL command.

ALTER CHANNEL




BATCHHB—(—integer—)
BATCHINT—(—integer—)
(2)
BATCHSZ—(—integer—)
CMDSCOPE(' ')
CMDSCOPE—(—qmgr-name—)
(3)
CMDSCOPE(*)
(3)
(4)
COMPHDR—(—NONE—)
SYSTEM
COMPMSG—(—NONE—)
RLE
ZLIBFAST
ZLIBHIGH
CONNAME—(—string—)
CONVERT—(—NO—)
YES
(4)
DEFCDISP—(—PRIVATE—)
FIXSHARED
SHARED
DESCR—(—string—)
DISCINT—(—integer—)
HBINT—(—integer—)
(2)
KAINT—(—integer—)
AUTO
LOCLADDR—(—string—)
LONGRTY—(—integer—)
LONGTMR—(—integer—)
MAXMSGL—(—integer—)
MCANAME—(—string—)
MCATYPE—(—PROCESS—)
THREAD
(5)
MCAUSER—(—string—)
MODENAME—(—string—)
(6)
MONCHL—(—QMGR—)
OFF
LOW
MEDIUM
HIGH
MSGDATA—(—string—)
(7)
MSGEXIT—(—string—)
(7)
NPMSPEED—(—FAST—)
NORMAL
(2)



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows and z/OS.

3 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

4 Valid only on z/OS.

5 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

6 Valid only if TRPTYPE is LU62.

7 You can specify more than one value on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS only.

8 Not valid on z/OS.

9 This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

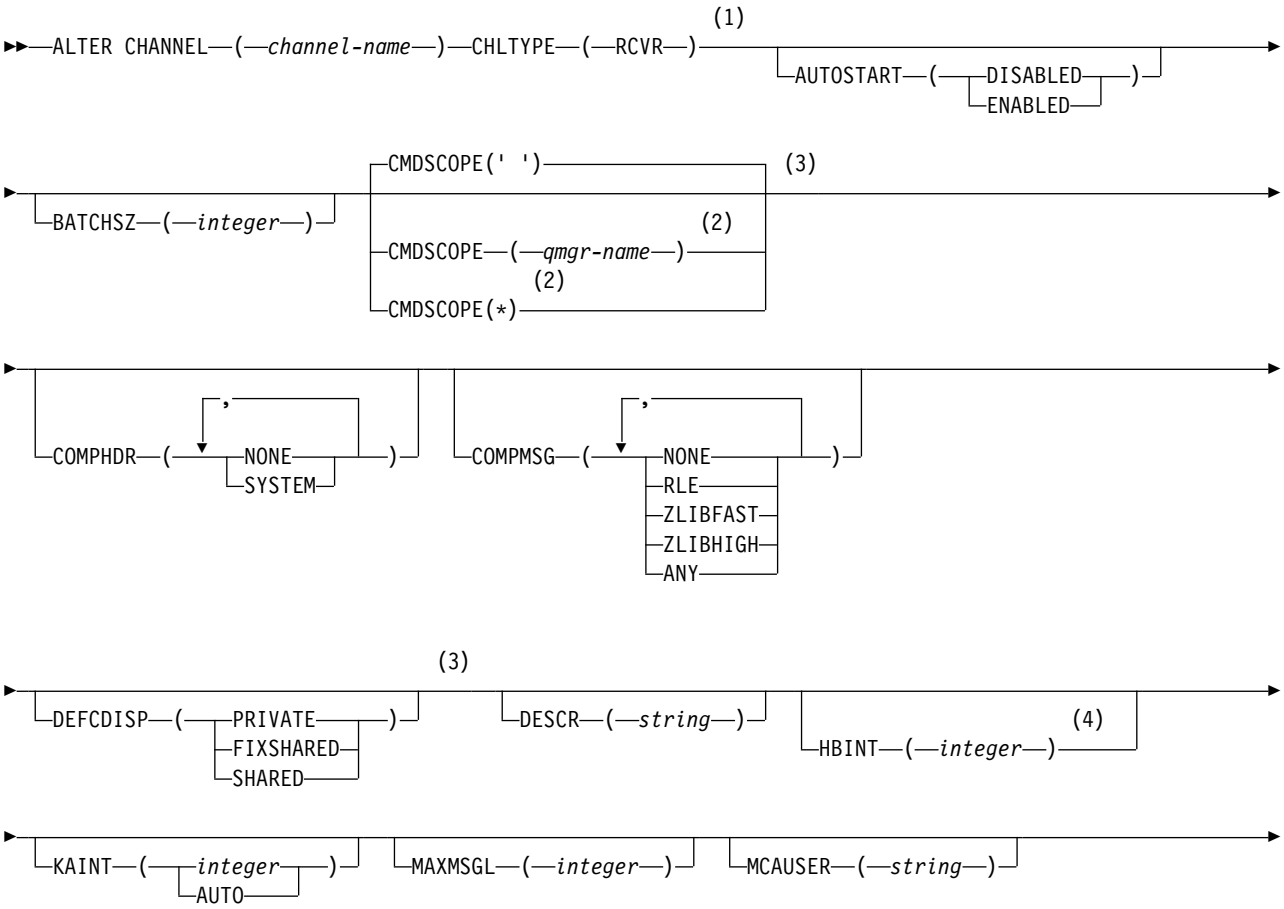
10 Valid only on Windows.

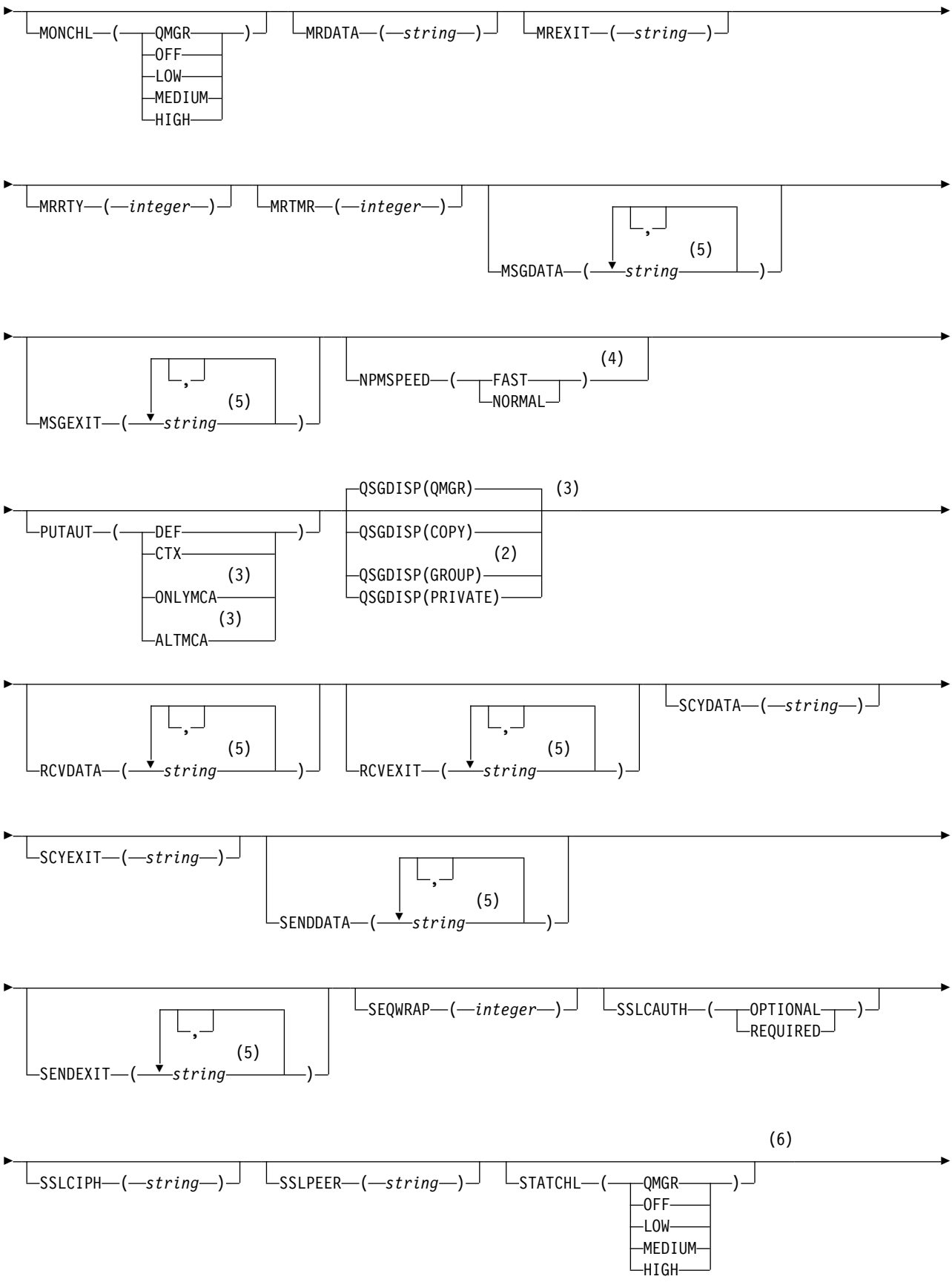
The parameters are described in "ALTER CHANNEL" on page 285.

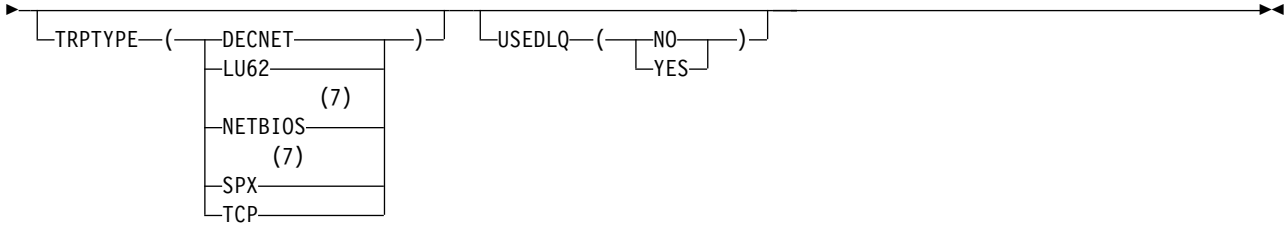
Receiver channel:

Syntax diagram for a receiver channel when using the ALTER CHANNEL command.

ALTER CHANNEL




MONCHL—(—QMGR—)
OFF
LOW
MEDIUM
HIGH
MRDATA—(—string—)
MREXIT—(—string—)
MRRTY—(—integer—)
MRTMR—(—integer—)
MSGDATA—(—string—)
,
(5)
MSGEXIT—(—string—)
,
(5)
NPMSPEED—(—FAST—)
NORMAL
(4)
PUTAUT—(—DEF—)
CTX
ONLYMCA
(3)
ALTMCA
(3)
QSGDISP(QMGR)
QSGDISP(COPY)
QSGDISP(GROUP)
(2)
QSGDISP(PRIVATE)
(3)
RCVDATA—(—string—)
,
(5)
RCVEXIT—(—string—)
,
(5)
SCYDATA—(—string—)
SCYEXIT—(—string—)
SENDDATA—(—string—)
,
(5)
SENDEXIT—(—string—)
,
(5)
SEQWRAP—(—integer—)
SSLCAUTH—(—OPTIONAL—)
REQUIRED
SSLCIPH—(—string—)
SSLPEER—(—string—)
STATCHL—(—QMGR—)
OFF
LOW
MEDIUM
HIGH
(6)



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

5 You can specify more than one value on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS only.

6 This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

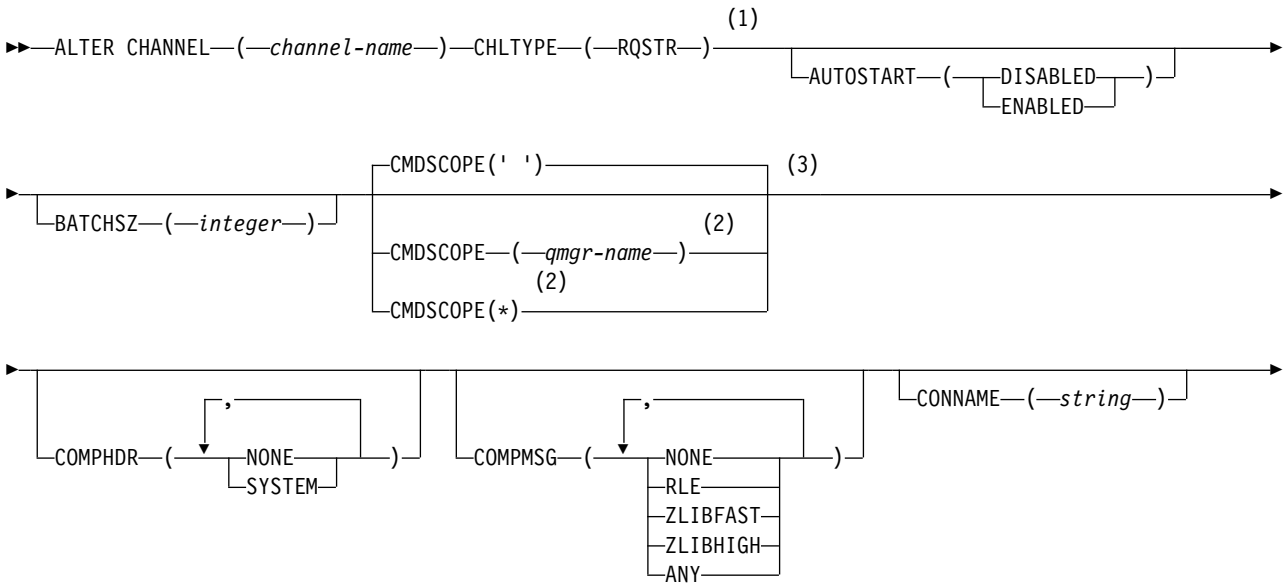
7 Valid only on Windows.

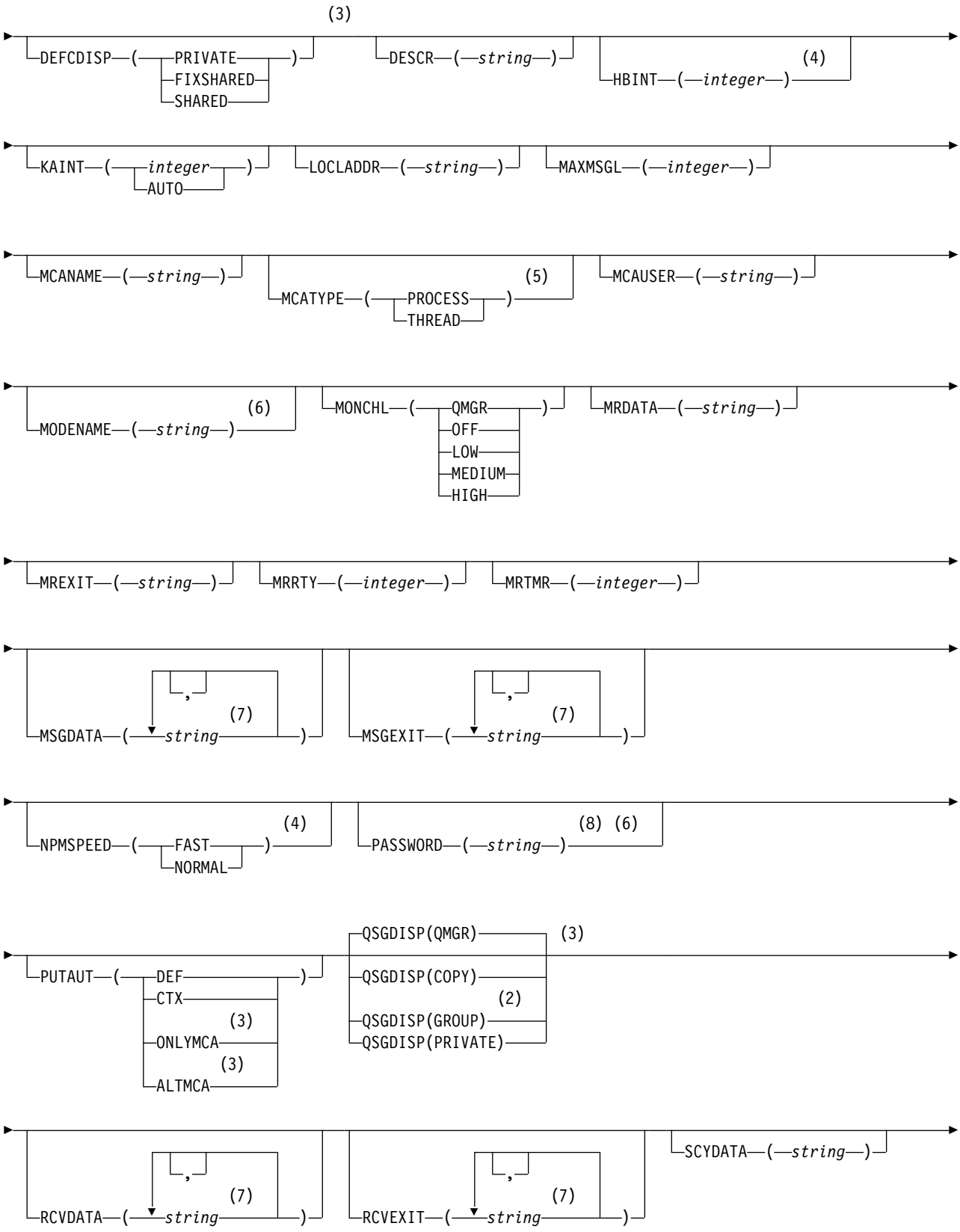
The parameters are described in "ALTER CHANNEL" on page 285.

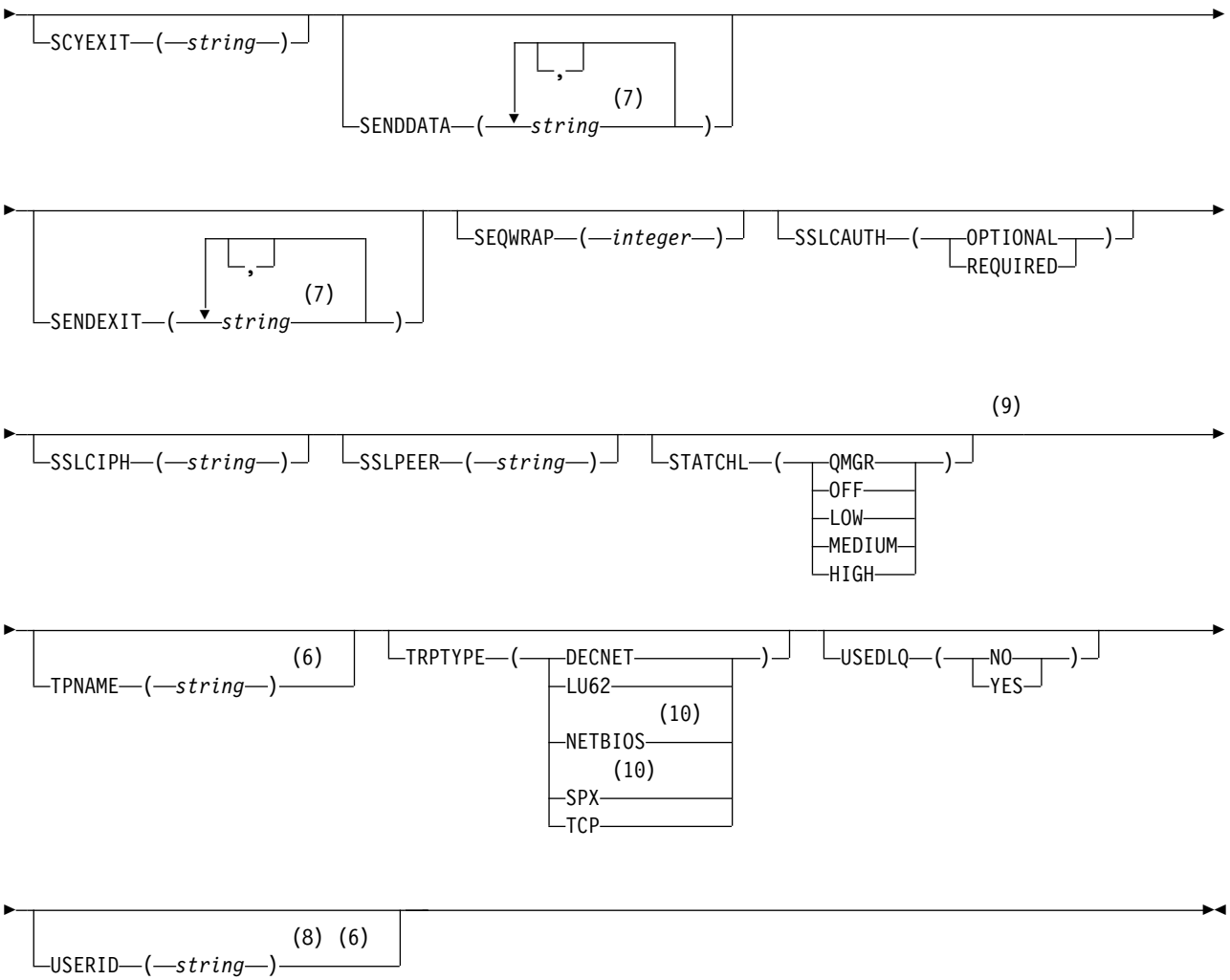
Requester channel:

Syntax diagram for a requester channel when using the ALTER CHANNEL command.

ALTER CHANNEL




DEFCDISP
PRIVATE
FIXSHARED
SHARED
(3)
DESCR
string
HBINT
integer
(4)
KAINT
integer
AUTO
LOCLADDR
string
MAXMSGL
integer
MCANAME
string
MCATYPE
PROCESS
THREAD
(5)
MCAUSER
string
MODENAME
string
(6)
MONCHL
QMGR
OFF
LOW
MEDIUM
HIGH
MRDATA
string
MREXIT
string
MRRTY
integer
MRTMR
integer
MSGDATA
string
(7)
MSGEXIT
string
(7)
NPMSPEED
FAST
NORMAL
(4)
PASSWORD
string
(8) (6)
PUTAUT
DEF
CTX
ONLYMCA
(3)
ALTMCA
(3)
QSGDISP(QMGR)
(3)
QSGDISP(COPY)
QSGDISP(GROUP)
(2)
QSGDISP(PRIVATE)
RCVDATA
string
(7)
RCVEXIT
string
(7)
SCYDATA
string



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

5 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

6 Valid only if TRPTYPE is LU62.

7 You can specify more than one value on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS only.

8 Not valid on z/OS.

9 This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

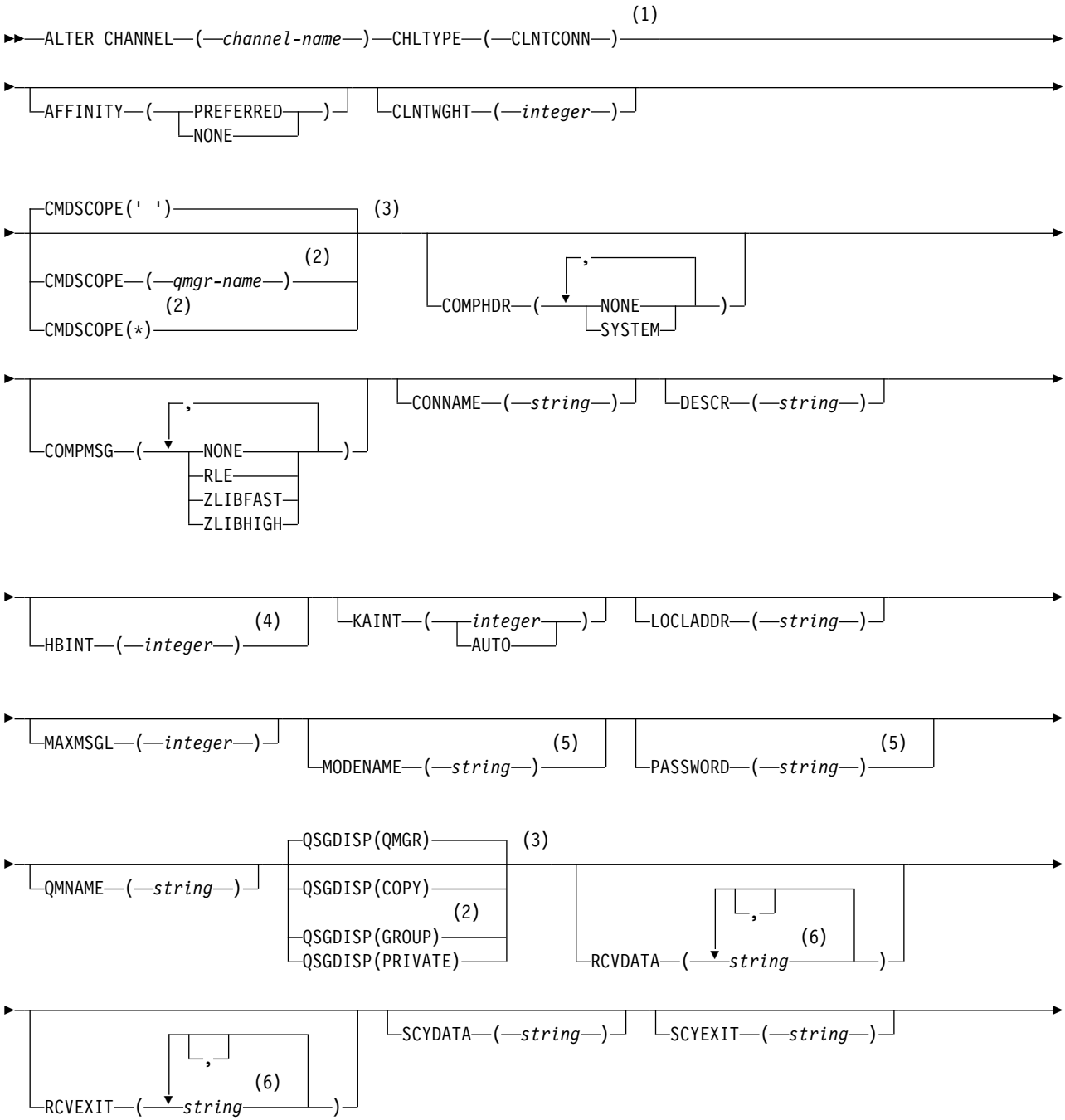
10 Valid only on Windows.

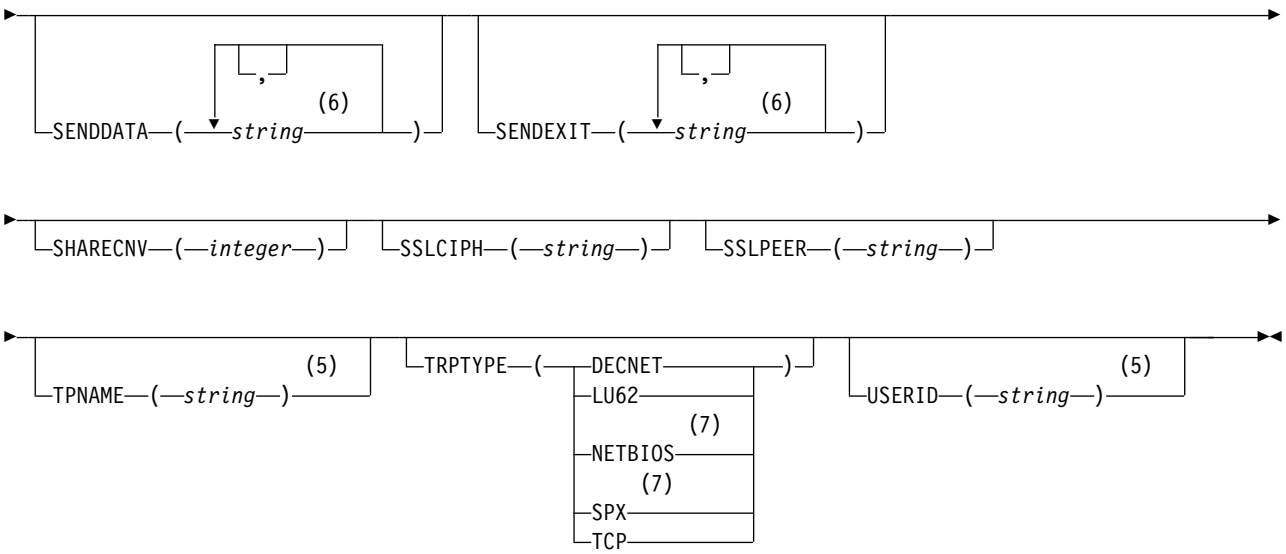
The parameters are described in "ALTER CHANNEL" on page 285.

Client-connection channel:

Syntax diagram for a client-connection channel when using the ALTER CHANNEL command.

ALTER CHANNEL





Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

5 Valid only if TRPTYPE is LU62.

6 You can specify more than one value on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS only.

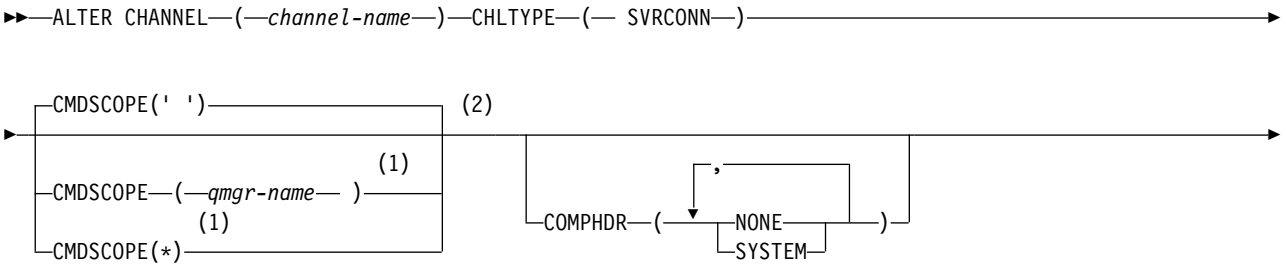
7 Valid only for clients to be run on DOS and Windows.

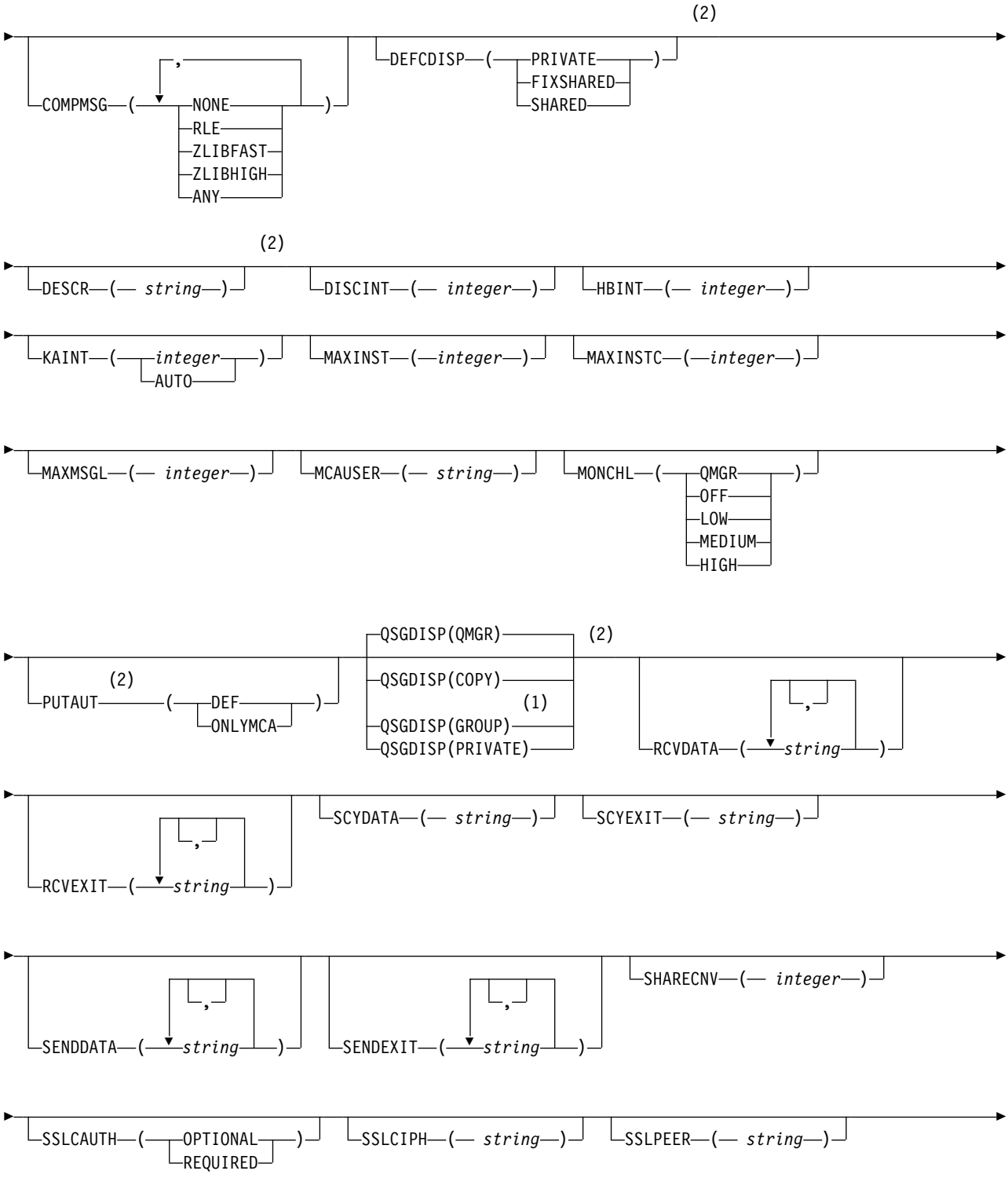
The parameters are described in "ALTER CHANNEL" on page 285.

Server-connection channel:

Syntax diagram for a server-connection channel when using the ALTER CHANNEL command.

ALTER CHANNEL




COMPMSG
NONE
RLE
ZLIBFAST
ZLIBHIGH
ANY
DEFCDISP
PRIVATE
FIXSHARED
SHARED
(2)
DESCR
string
DISCINT
integer
HBINT
integer
KAINT
integer
AUTO
MAXINST
integer
MAXINSTC
integer
MAXMSGL
integer
MCAUSER
string
MONCHL
QMGR
OFF
LOW
MEDIUM
HIGH
PUTAUT
DEF
ONLYMCA
QSGDISP(QMGR)
QSGDISP(COPY)
QSGDISP(GROUP)
QSGDISP(PRIVATE)
(1)
RCVDATA
string
RCVEXIT
string
SCYDATA
string
SCYEXIT
string
SENDDATA
string
SENDEXIT
string
SHARECNV
integer
SSLCAUTH
OPTIONAL
REQUIRED
SSLCIPH
string
SSLPEER
string

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

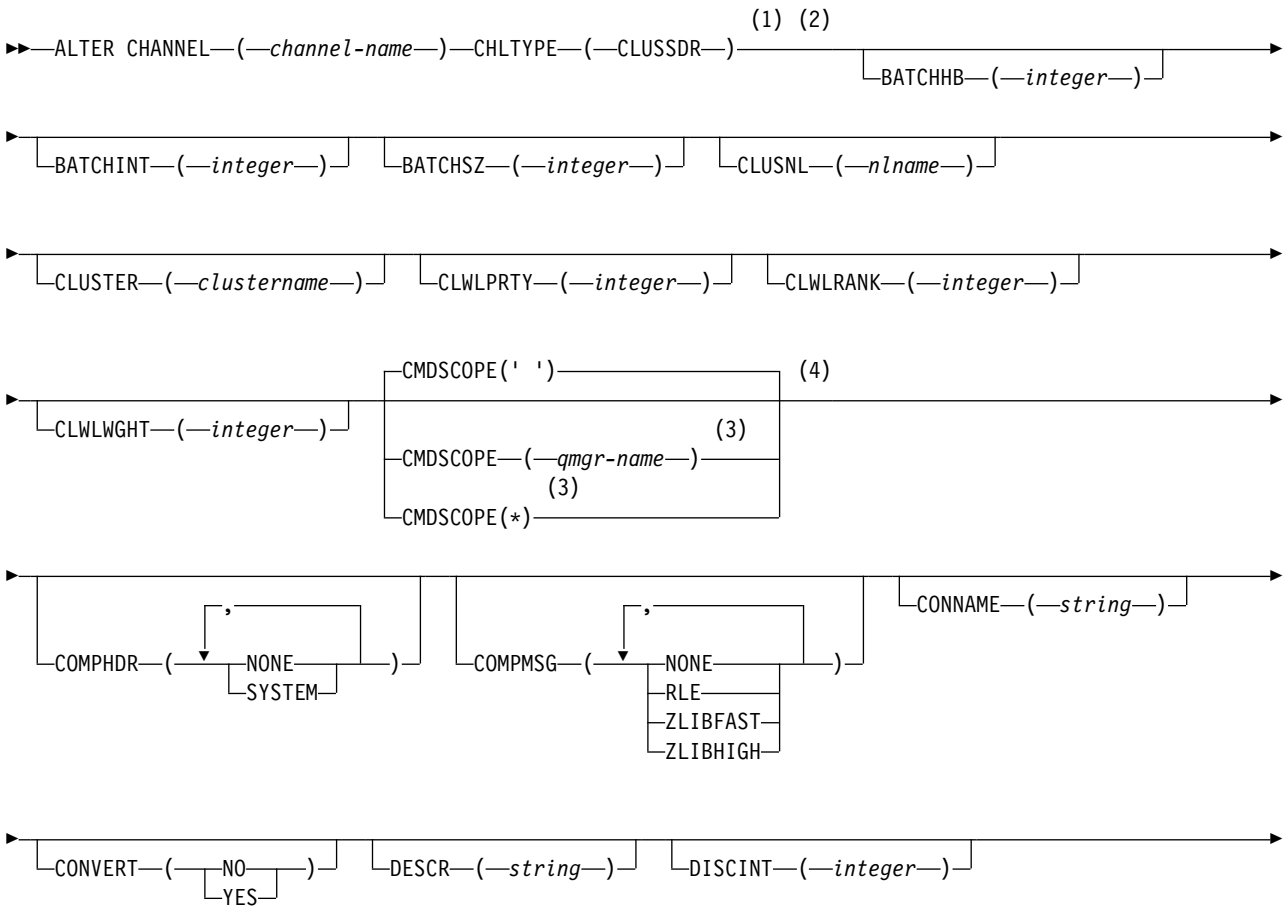
3 Valid only for clients to be run on Windows.

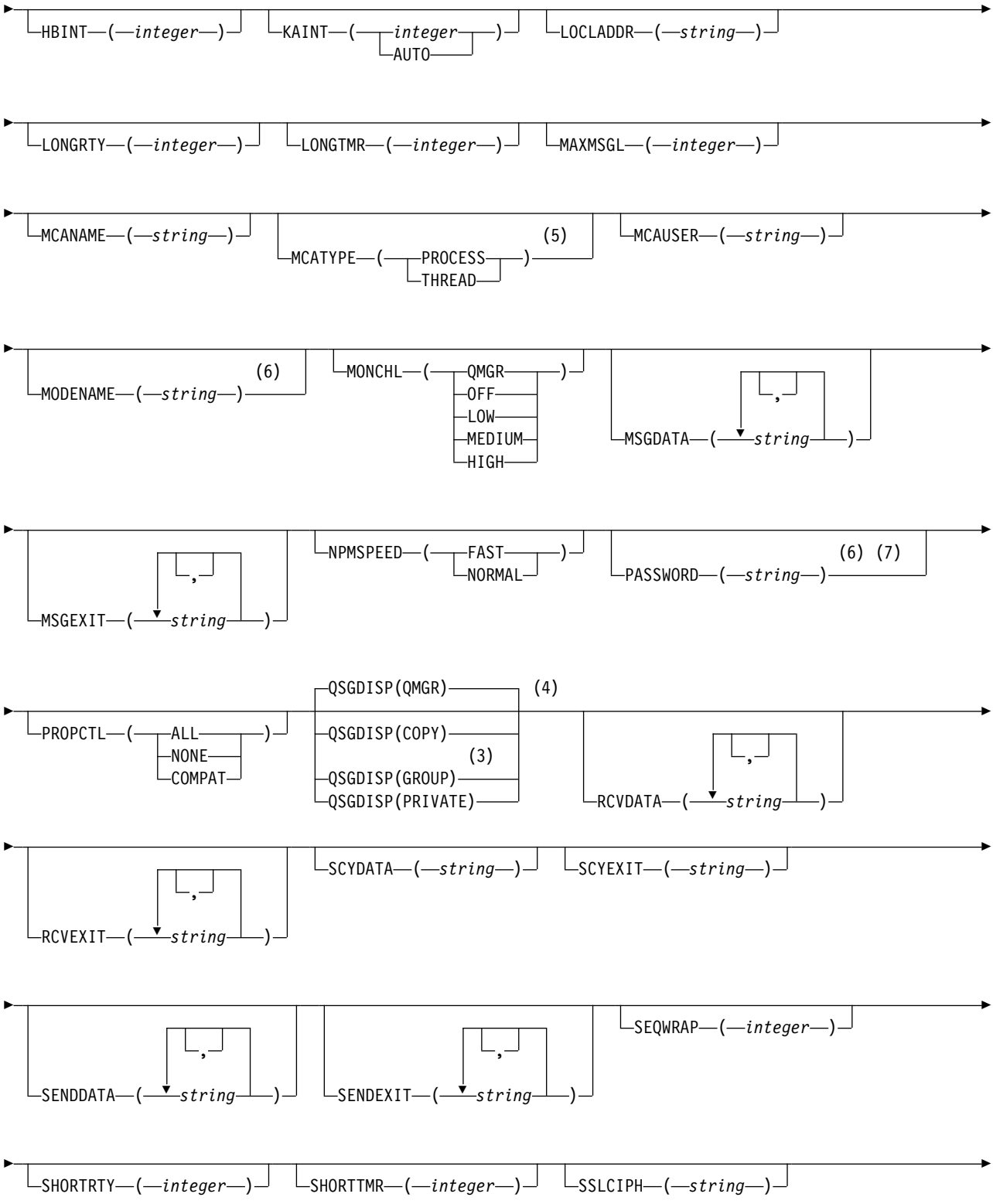
The parameters are described in "ALTER CHANNEL" on page 285.

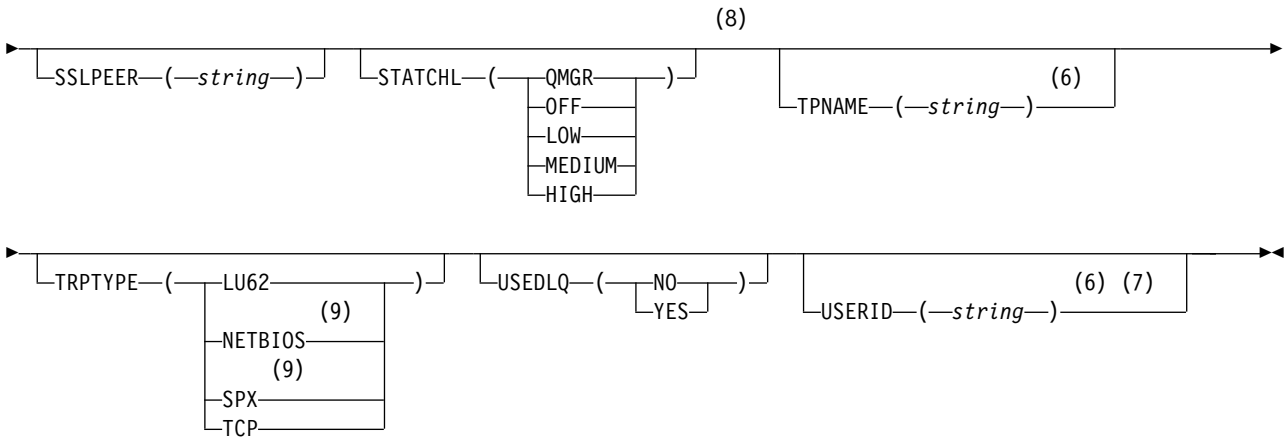
Cluster-sender channel:

Syntax diagram for a cluster-sender channel when using the ALTER CHANNEL command.

ALTER CHANNEL




HBINT—(—integer—)
KAINT—(—integer—)
AUTO
LOCLADDR—(—string—)
LONGRTY—(—integer—)
LONGTMR—(—integer—)
MAXMSGL—(—integer—)
MCANAME—(—string—)
MCATYPE—(—PROCESS—)
THREAD
(5)
MCAUSER—(—string—)
MODENAME—(—string—)
(6)
MONCHL—(—QMGR—)
OFF
LOW
MEDIUM
HIGH
MSGDATA—(—string—)
,
MSGEXIT—(—string—)
NPMSPEED—(—FAST—)
NORMAL
PASSWORD—(—string—)
(6) (7)
PROPCTL—(—ALL—)
NONE
COMPAT
QSGDISP(QMGR)
QSGDISP(COPY)
QSGDISP(GROUP)
QSGDISP(PRIVATE)
(4)
(3)
RCVDATA—(—string—)
RCVEXIT—(—string—)
SCYDATA—(—string—)
SCYEXIT—(—string—)
SENDDATA—(—string—)
SENDEXIT—(—string—)
SEQWRAP—(—integer—)
SHORTRTY—(—integer—)
SHORTTMR—(—integer—)
SSLCIPH—(—string—)



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

3 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

4 Valid only on z/OS.

5 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

6 Valid only if TRPTYPE is LU62.

7 Not valid on z/OS.

8 This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

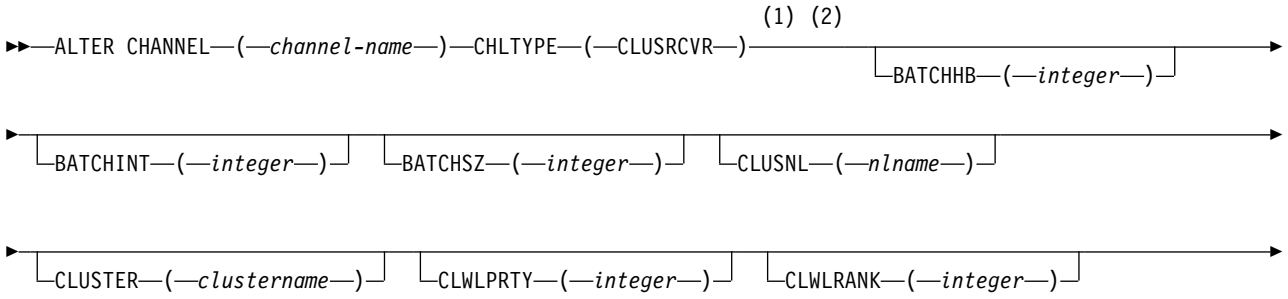
9 Valid only Windows.

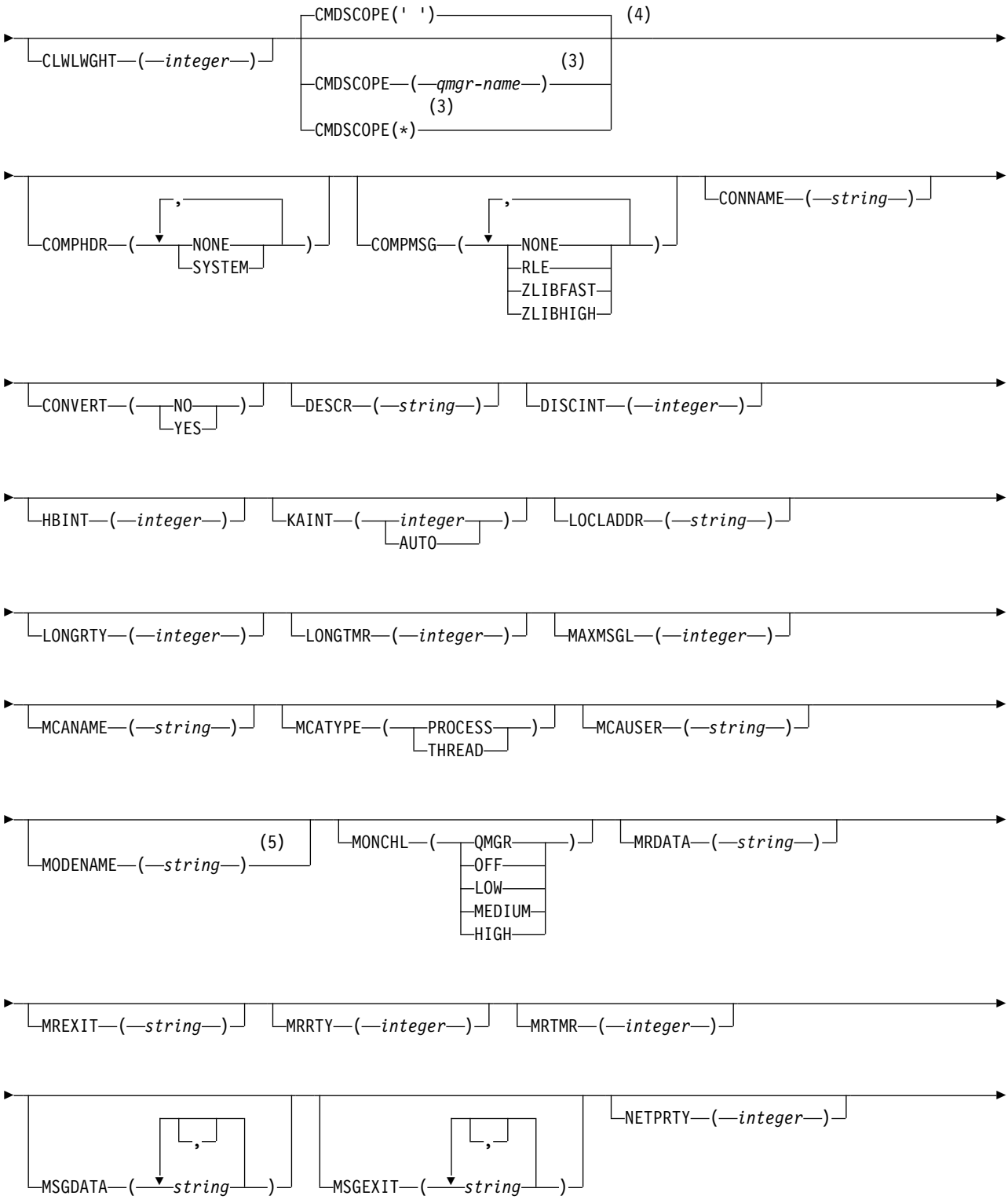
The parameters are described in "ALTER CHANNEL" on page 285.

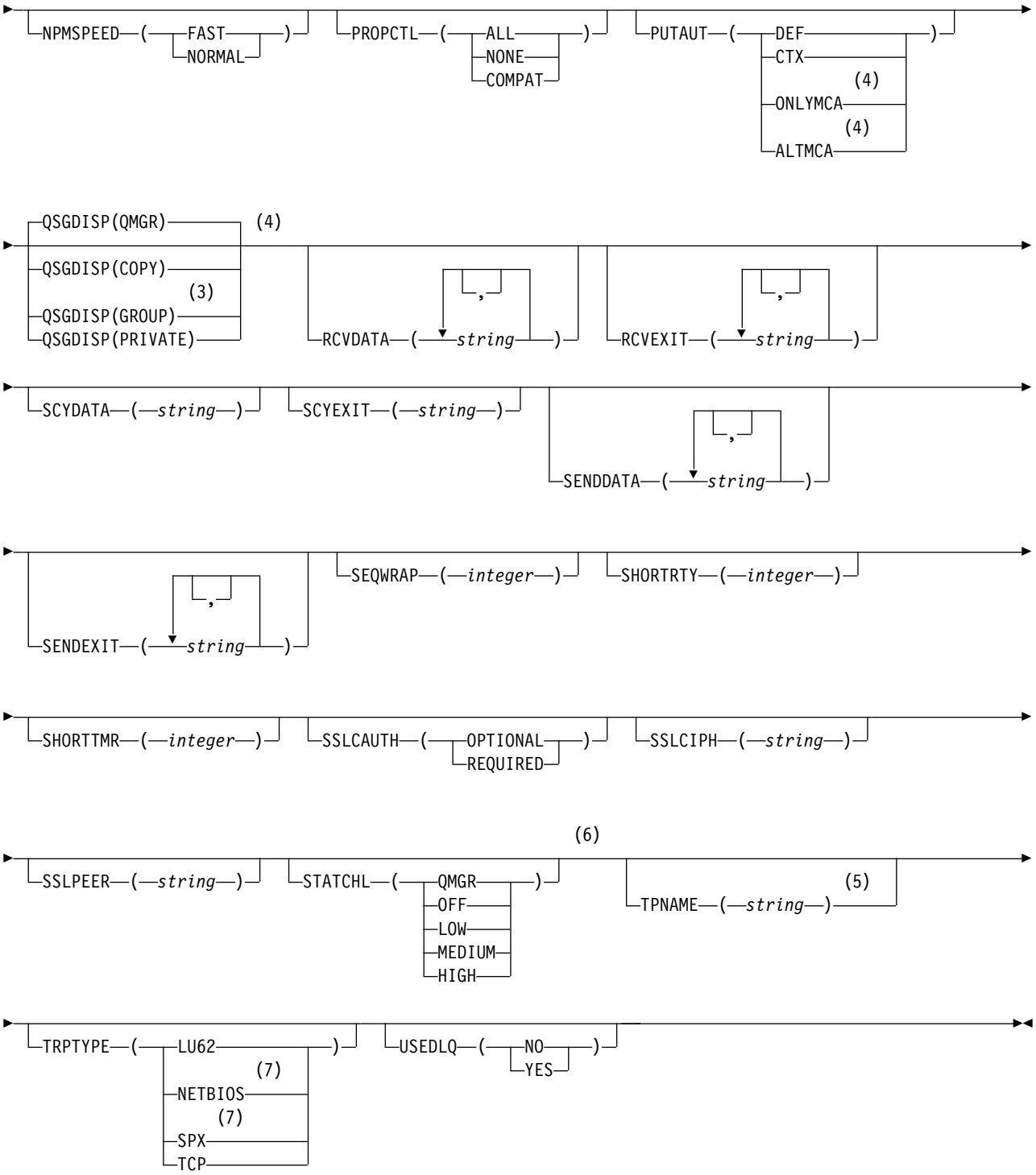
Cluster-receiver channel:

Syntax diagram for a cluster-receiver channel when using the ALTER CHANNEL command.

ALTER CHANNEL




CMDSCOPE(' ') (4)
CLWLWGHT (integer)
CMDSCOPE (qmgr-name) (3)
CMDSCOPE(*) (3)
COMPHDR (NONE SYSTEM)
COMPMSG (NONE RLE ZLIBFAST ZLIBHIGH)
CONNAME (string)
CONVERT (NO YES)
DESCR (string)
DISCINT (integer)
HBINT (integer)
KAINT (integer AUTO)
LOCLADDR (string)
LONGRTY (integer)
LONGTMR (integer)
MAXMSGL (integer)
MCANAME (string)
MCATYPE (PROCESS THREAD)
MCAUSER (string)
MODENAME (string) (5)
MONCHL (QMGR OFF LOW MEDIUM HIGH)
MRDATA (string)
MREXIT (string)
MRRTY (integer)
MRTMR (integer)
MSGDATA (string)
MSGEXIT (string)
NETPRTY (integer)



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

3 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

4 Valid only on z/OS.

5 Valid only if TRPTYPE is LU62.

6 This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

7 Valid only on Windows.

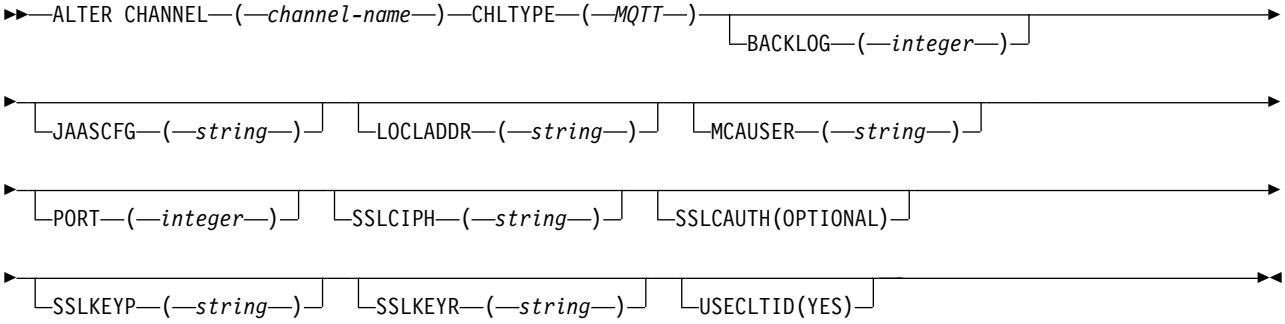
The parameters are described in "ALTER CHANNEL" on page 285.

ALTER CHANNEL (MQTT):

Syntax diagram for a telemetry channel when using the ALTER CHANNEL command. This is separate from the regular ALTER CHANNEL syntax diagram and parameter descriptions.

IBM i	UNIX and Linux	Windows	z/OS
	✓	✓	

Note: For the telemetry server, AIX is the only supported UNIX platform.



Parameter descriptions for ALTER CHANNEL (MQTT)

(channel-name)
The name of the new channel definition.

The name must not be the same as any existing channel defined on this queue manager (unless REPLACE or ALTER is specified).

The maximum length of the string is 20 characters, and the string must contain only valid characters; see Rules for naming IBM WebSphere MQ objects.

CHLTYPE
Channel type. This parameter is required.

MQTT Telemetry channel

BACKLOG(*integer***)**
The number of outstanding connection requests that the telemetry channel can support at any one time. When the backlog limit is reached, any further clients trying to connect will be refused connection until the current backlog is processed.

The value is in the range 0 - 999999999.

The default value is 4096.

JAASCFG(*string***)**
The file path of the JAAS configuration.

LOCLADDR(*string***)**
LOCLADDR is the local communications address for the channel. Use this parameter if you want

a channel to use a particular IP address, port, or port range for outbound communications. LOCLADDR might be useful in recovery scenarios where a channel is restarted on a different TCP/IP stack. LOCLADDR is also useful to force a channel to use an IPv4 or IPv6 stack on a dual-stack system. You can also use LOCLADDR to force a channel to use a dual-mode stack on a single-stack system.

This parameter is valid only for channels with a transport type (TRPTYPE) of TCP. If TRPTYPE is not TCP, the data is ignored and no error message is issued.

The value is the optional IP address, and optional port or port range used for outbound TCP/IP communications. The format for this information is as follows:
Table 73 on page 452 shows how the LOCLADDR parameter can be used:

```
LOCLADDR([ip-addr][(low-port[,high-port])][,[ip-addr][(low-port[,high-port])]])
```

The maximum length of LOCLADDR, including multiple addresses, is MQ_LOCAL_ADDRESS_LENGTH.

If you omit LOCLADDR, a local address is automatically allocated.

Note, that you can set LOCLADDR for a C client using the Client Channel Definition Table (CCDT).

All the parameters are optional. Omitting the ip-addr part of the address is useful to enable the configuration of a fixed port number for an IP firewall. Omitting the port number is useful to select a particular network adapter without having the identify a unique local port number. The TCP/IP stack generates a unique port number.

Specify *[,[ip-addr][(low-port[,high-port])]]* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *[,[ip-addr][(low-port[,high-port])]]* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

ip-addr
ip-addr is specified in one of three forms:

IPv4 dotted decimal
For example 192.0.2.1

IPv6 hexadecimal notation
For example 2001:DB8:0:0:0:0:0:0

Alphanumeric host name form
For example WWW.EXAMPLE.COM

low-port and high-port
low-port and high-port are port numbers enclosed in parentheses.

Table 68. Examples of how the LOCLADDR parameter can be used

LOCLADDR	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98, 9.20.4.99	Channel binds to either IP address. The address might be two network adapters on one server, or a different network adapter on two different servers in a multi-instance configuration.
9.20.4.98(1000)	Channel binds to this address and port 1000 locally
9.20.4.98(1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to port in range 1000 - 2000 locally

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, CLUSRCVR, or MQTT.

On CLUSSDR channels, the IP address and port to which the outbound channel binds, is a combination of fields. It is a concatenation of the IP address, as defined in the LOCLADDR parameter, and the port range from the cluster cache. If there is no port range in the cache, the port range defined in the LOCLADDR parameter is used. This port range does not apply to z/OS.

Even though this parameter is similar in form to CONNAME, it must not be confused with it. The LOCLADDR parameter specifies the characteristics of the local communications, whereas the CONNAME parameter specifies how to reach a remote queue manager.

When a channel is started, the values specified for CONNAME and LOCLADDR determine the IP stack to be used for communication; see Table 3 and Local Address (LOCLADDR) .

If the TCP/IP stack for the local address is not installed or configured, the channel does not start and an exception message is generated. The message indicates that the connect() request specifies an interface address that is not known on the default IP stack. To direct the connect() request to the alternative stack, specify the **LOCLADDR** parameter in the channel definition as either an interface on the alternative stack, or a DNS host name. The same specification also works for listeners that might not use the default stack. To find the value to code for **LOCLADDR**, run the **NETSTAT HOME** command on the IP stacks that you want to use as alternatives.

For channels with a channel type (CHLTYPE) of MQTT the usage of this parameter is slightly different. Specifically, a telemetry channel (MQTT) **LOCLADDR** parameter expects only an IPv4 or IPv6 IP address, or a valid host name as a string. This string must not contain a port number or port range. If an IP address is entered, only the address format is validated. The IP address itself is not validated.

Table 69. How the IP stack to be used for communication is determined

Protocols supported	CONNAME	LOCLADDR	Action of channel
IPv4 only	IPv4 address[1]		Channel binds to IPv4 stack
	IPv6 address[2]		Channel fails to resolve CONNAME
	IPv4 and 6 host name[3]		Channel binds to IPv4 stack
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	Any address[4]	IPv6 address	Channel fails to resolve LOCLADDR
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv4 stack

Table 69. How the IP stack to be used for communication is determined (continued)

Protocols supported	CONNAME	LOCLADDR	Action of channel
IPv4 and IPv6	IPv4 address		Channel binds to IPv4 stack
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to stack determined by IPADDRV
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	IPv4 address	IPv6 address	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv6 address	Channel binds IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to stack determined by IPADDRV
IPv6 only	IPv4 address		Channel maps CONNAME to IPv6[5]
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to IPv6 stack
	Any address	IPv4 address	Channel fails to resolve LOCLADDR
	IPv4 address	IPv6 address	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv6 address	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds to IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv6 stack

Notes:

1. IPv4 address. An IPv4 host name that resolves only to an IPv4 network address or a specific dotted notation IPv4 address, for example `1.2.3.4`. This note applies to all occurrences of 'IPv4 address' in this table.
2. IPv6 address. An IPv6 host name that resolves only to an IPv6 network address or a specific hexadecimal notation IPv6 address, for example `4321:54bc`. This note applies to all occurrences of 'IPv6 address' in this table.
3. IPv4 and 6 host name. A host name that resolves to both IPv4 and IPv6 network addresses. This note applies to all occurrences of 'IPv4 and 6 host name' in this table.
4. Any address. IPv4 address, IPv6 address, or IPv4 and 6 host name. This note applies to all occurrences of 'Any address' in this table.
5. Maps IPv4 CONNAME to IPv4 mapped IPv6 address. IPv6 stack implementations that do not support IPv4 mapped IPv6 addressing fail to resolve the CONNAME. Mapped addresses might require protocol translators in order to be used. The use of mapped addresses is not recommended.

MCAUSER(*string***)**

Message channel agent user identifier.

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication

records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL). For more details, see Channel authentication records.

This parameter interacts with PUTAUT, see the definition of that parameter for more information.

If it is nonblank, it is the user identifier that is to be used by the message channel agent for authorization to access IBM WebSphere MQ resources, including (if PUTAUT is DEF) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

The default user identifier is derived from the user ID that started the receiving channel. The possible values are:

- For TCP/IP, the user ID from the `inetd.conf` entry, or the user that started the listener.
- For SNA, the user ID from the SNA server entry or, in the absence of this user ID the incoming attach request, or the user that started the listener.
- For NetBIOS or SPX, the user ID that started the listener.

The maximum length of the string is 64 characters on Windows and 12 characters on other platforms. On Windows, you can optionally qualify a user identifier with the domain name in the format `user@domain`.

PORT(*integer***)**

The port number for TCP/IP. This parameter is the port number on which the listener is to stop listening. It is valid only if the transmission protocol is TCP/IP.

The PORT parameter accepts a value of zero. This value causes an available port to be assigned to the channel.

SSLCAUTH

Defines whether IBM WebSphere MQ requires a certificate from the SSL client. The initiating end of the channel acts as the SSL client, so this parameter applies to the end of the channel that receives the initiation flow, which acts as the SSL server.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, SVRCONN, CLUSRCVR, SVR, RQSTR, or MQTT.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

REQUIRED

IBM WebSphere MQ requires and validates a certificate from the SSL client.

OPTIONAL

The peer SSL client system might still send a certificate. If it does, the contents of this certificate are validated as normal.

SSLCIPH*(string)*

When SSLCIPH is used with a telemetry channel, it means “SSL Cipher Suite”. The SSL cipher suite is the one supported by the JVM that is running the telemetry (MQXR) service. If the SSLCIPH parameter is blank, no attempt is made to use SSL on the channel.

Here is an alphabetic list of the SSL cipher suites that are currently supported:

- `SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
- `SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
- `SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`
- `SSL_DH_anon_WITH_AES_128_CBC_SHA`
- `SSL_DH_anon_WITH_DES_CBC_SHA`
- `SSL_DH_anon_WITH_RC4_128_MD5`

- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_RC4_128_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- SSL_KRB5_EXPORT_WITH_RC4_40_MD5
- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA
- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 If you plan to use SHA-2 cipher suites, see System requirements for using SHA-2 cipher suites with MQTT channels.

SSLKEYP(*string***)**
The store for digital certificates and their associated private keys. If you do not specify a key file, SSL is not used.

SSLKEYR(*string***)**
The password for the key repository. If no passphrase is entered, then unencrypted connections must be used.

USECLTID

Decide whether you want to use the MQTT client ID for the new connection as the IBM WebSphere MQ user ID for that connection. If this property is specified, the user name supplied by the client is ignored.

Related reference:

"DEFINE CHANNEL (MQTT)" on page 489

Syntax diagram for a telemetry channel when using the **DEFINE CHANNEL** command.

Related information:

Telemetry channel configuration for MQTT client authentication using SSL

Telemetry channel configuration for channel authentication using SSL

CipherSpecs and CipherSuites

V 7.5.0.2 System requirements for using SHA-2 cipher suites with MQTT channels

ALTER COMMINFO:

Use the MQSC command ALTER COMMINFO to alter the parameters of a communication information object.

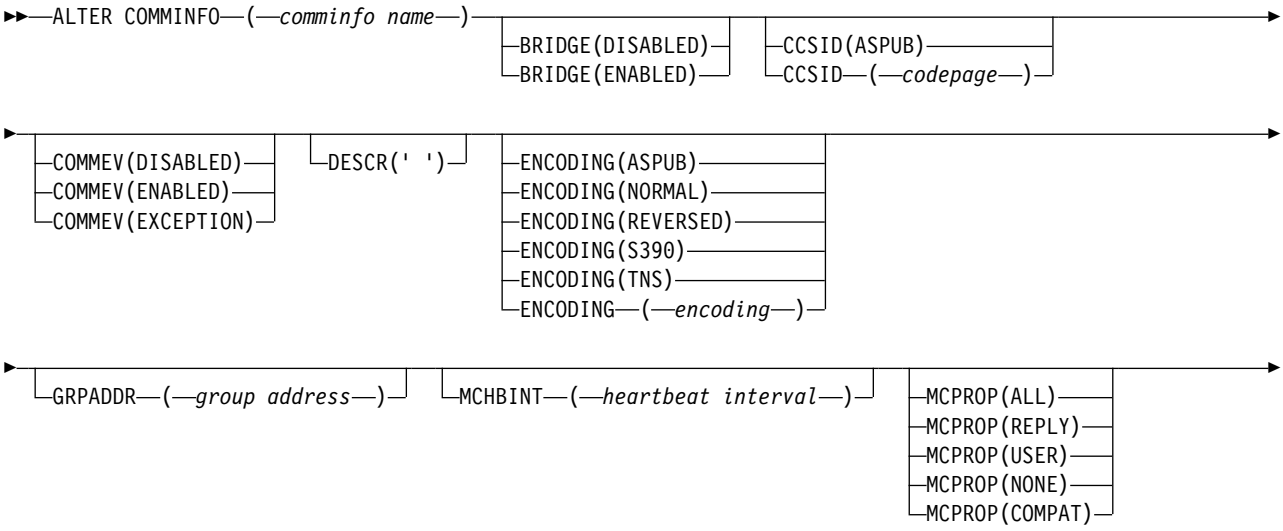
UNIX and Linux	Windows
✓	✓

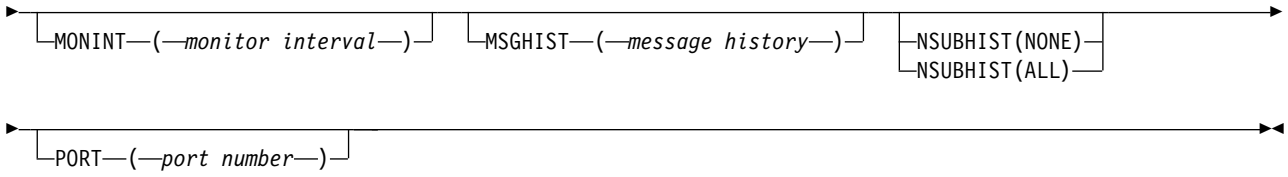
Parameters not specified in the ALTER COMMINFO command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Parameter descriptions for ALTER COMMINFO" on page 342

Synonym: ALT COMMINFO

ALTER COMMINFO





Parameter descriptions for ALTER COMMINFO

(comminfo name)
Name of the communications information object. This parameter is required.

The name must not be the same as any other communications information object name currently defined on this queue manager. See Rules for naming IBM WebSphere MQ objects .

BRIDGE
Controls whether publications from applications not using Multicast are bridged to applications using Multicast. Bridging does not apply to topics that are marked as **MCAST(ONLY)**. As these topics can only be Multicast traffic, it is not applicable to bridge to the queue's publish/subscribe domain.

DISABLED
Publications from applications not using Multicast are not bridged to applications that do use Multicast.

ENABLED
Publications from applications not using Multicast are bridged to applications that do use Multicast.

CCSID(*integer***)**
The coded character set identifier that messages are transmitted on. Specify a value in the range 1 through 65535.

The CCSID must specify a value that is defined for use on your platform, and use a character set that is appropriate to the queue manager's platform. If you use this parameter to change the CCSID, applications that are running when the change is applied continue to use the original CCSID therefore you must stop and restart all running applications before you continue. Running applications include the command server and channel programs. Stop and restart all running applications, stop and restart the queue manager after changing this parameter.

The CCSID can also be set to ASPUB which means that the coded character set is taken from that supplied in the published message.

COMMEV
Controls whether event messages are generated for Multicast handles that are created using this COMMINFO object. Events will only be generated if they are enabled using the **MONINT** parameter.

DISABLED
Publications from applications not using Multicast are not bridged to applications that do use Multicast.

ENABLED
Publications from applications not using Multicast are bridged to applications that do use Multicast.

EXCEPTION
Event messages are written if the message reliability is below the reliability threshold The reliability threshold is set to 90 by default.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the communication information object when an operator issues the DISPLAY COMMINFO command (see "DISPLAY COMMINFO" on page 637).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

ENCODING
The encoding that the messages are transmitted in.

ASPUB
The encoding of the message is taken from that supplied in the published message.

NORMAL

REVERSED

S390

TNS

encoding

GRPADDR
The group IP address or DNS name.

It is the responsibility of the administrator to manage the group addresses. It is possible for all multicast clients to use the same group address for every topic; only the messages that match outstanding subscriptions on the client are delivered. Using the same group address can be inefficient because every client has to examine and process every multicast packet in the network. It is more efficient to allocate different IP group addresses to different topics or sets of topics, but this allocation requires careful management, especially if other non-MQ multicast applications are in use on the network.

MCHBINT
The heartbeat interval is measured in milliseconds, and specifies the frequency at which the transmitter notifies any receivers that there is no further data available.

MCPROP
The multicast properties control how many of the MQMD properties and user properties flow with the message.

All
All user properties and all the fields of the MQMD are transported.

Reply
Only user properties, and MQMD fields that deal with replying to the messages, are transmitted. These properties are:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

User
Only the user properties are transmitted.

NONE
No user properties or MQMD fields are transmitted.

COMPAT
This value causes the transmission of the message to be done in a compatible mode to RMM allowing some inter-operation with the current XMS applications and Broker RMM applications.

MONINT(*integer***)**
How frequently, in seconds, that monitoring information is updated. If events messages are enabled, this parameter also controls how frequently event messages are generated about the status of the Multicast handles created using this COMMINFO object.

A value of 0 means that there is no monitoring.

MSGHIST
The maximum message history is the amount of message history that is kept by the system to handle retransmissions in the case of NACKs (negative acknowledgments).

A value of `0` gives the least level of reliability.

NSUBHIST
The new subscriber history controls whether a subscriber joining a publication stream receives as much data as is currently available, or receives only publications made from the time of the subscription.

NONE
A value of `NONE` causes the transmitter to transmit only publication made from the time of the subscription.

ALL
A value of `ALL` causes the transmitter to retransmit as much history of the topic as is known. In some circumstances, this retransmission can give a similar behavior to retained publications.

Note: Using the value of `ALL` might have a detrimental effect on performance if there is a large topic history because all the topic history is retransmitted.

PORT(*integer***)**
The port number to transmit on.

ALTER LISTENER:

Use MQSC command ALTER LISTENER to alter the parameters of an existing WebSphere MQ listener definition. If the listener is already running, any changes you make to its definition are effective only after the next time that the listener is started.

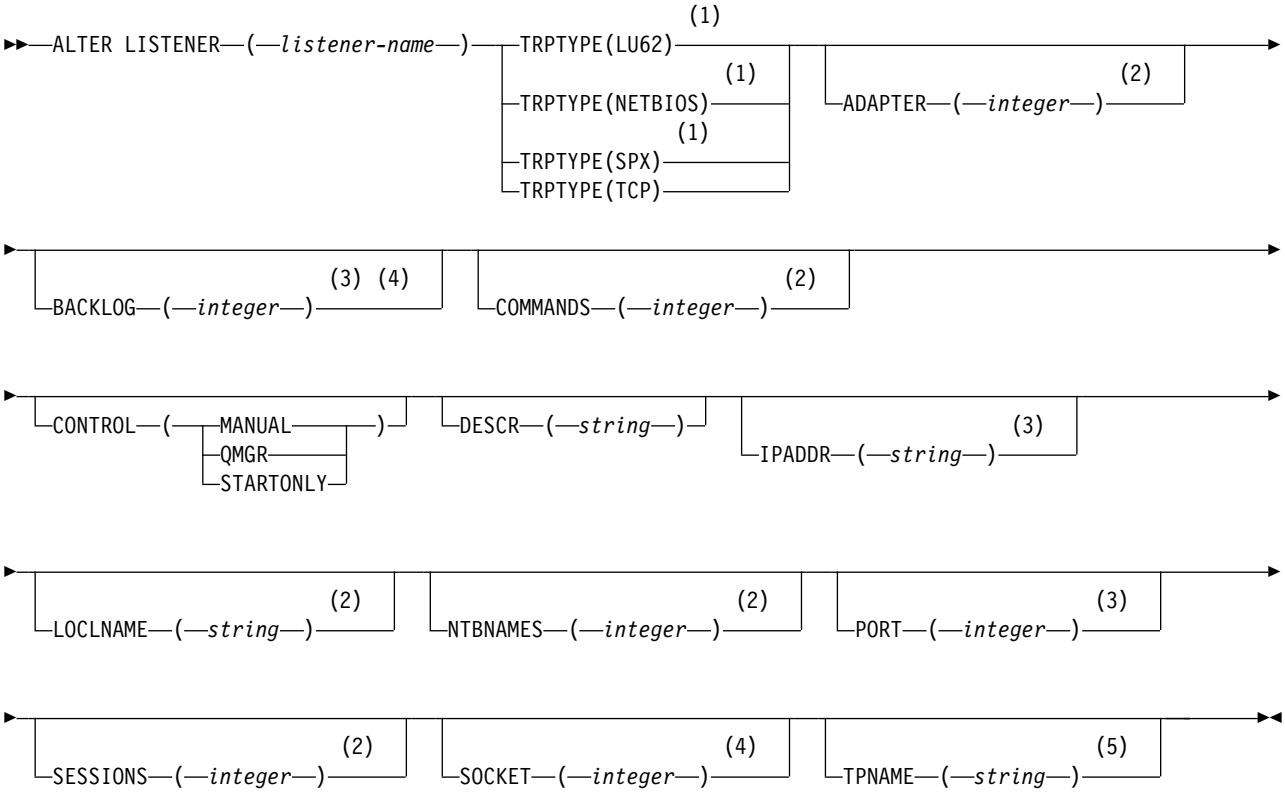
UNIX and Linux	Windows
✓	✓

Parameters not specified in the ALTER LISTENER command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Parameter descriptions for ALTER LISTENER" on page 345

Synonym: ALT LSTR

ALTER LISTENER



Notes:

1 Valid only on Windows.

2 Valid only on Windows when TRPTYPE is NETBIOS.

3 Valid when TRPTYPE is TCP.

4 Valid on Windows when TRPTYPE is SPX.

5 Valid only on Windows when TRPTYPE is LU62.

Parameter descriptions for ALTER LISTENER

(listener-name)
Name of the WebSphere MQ listener definition (see Rules for naming IBM WebSphere MQ objects). This is required.

The name must not be the same as any other listener definition currently defined on this queue manager (unless REPLACE is specified).

ADAPTER(*integer***)**
The adapter number on which NetBIOS listens. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

BACKLOG(*integer***)**
The number of concurrent connection requests that the listener supports.

COMMANDS(*integer***)**
The number of commands that the listener can use. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

CONTROL(*string***)**
Specifies how the listener is to be started and stopped.:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the listener when an operator issues the DISPLAY LISTENER command (see "DISPLAY LISTENER" on page 654).

It should contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

IPADDR(*string***)**
IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form. If you do not specify a value for this parameter, the listener listens on all configured IPv4 and IPv6 stacks.

LIKE(*listener-name***)**
The name of a listener, with parameters that are used to model this definition.

This parameter applies only to the DEFINE LISTENER command.

If this field is not filled in, and you do not complete the parameter fields related to the command, the values are taken from the default definition for listeners on this queue manager. This is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.LISTENER)
```

A default listener is provided but it can be altered by the installation of the default values required. See Rules for naming IBM WebSphere MQ objects .

LOCLNAME(*string***)**
The NetBIOS local name that the listener uses. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

NTBNAMES(*integer***)**
The number of names that the listener can use. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

PORT(*integer***)**
The port number for TCP/IP. This is valid only when TRPTYPE is TCP. It must not exceed 65535.

SESSIONS(*integer***)**
The number of sessions that the listener can use. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

SOCKET(*integer***)**
The SPX socket on which to listen. This is valid only if TRPTYPE is SPX.

TPNAME(*string***)**
The LU 6.2 transaction program name (maximum length 64 characters). This parameter is valid only on Windows when TRPTYPE is LU62.

TRPTYPE(*string***)**
The transmission protocol to be used:

LU62
SNA LU 6.2. This is valid only on Windows.

NETBIOS
NetBIOS. This is valid only on Windows.

SPX
Sequenced packet exchange. This is valid only on Windows.

TCP
TCP/IP.

ALTER NAMELIST:

Use the MQSC command ALTER NAMELIST to alter a list of names. This list is most commonly a list of cluster names or queue names.

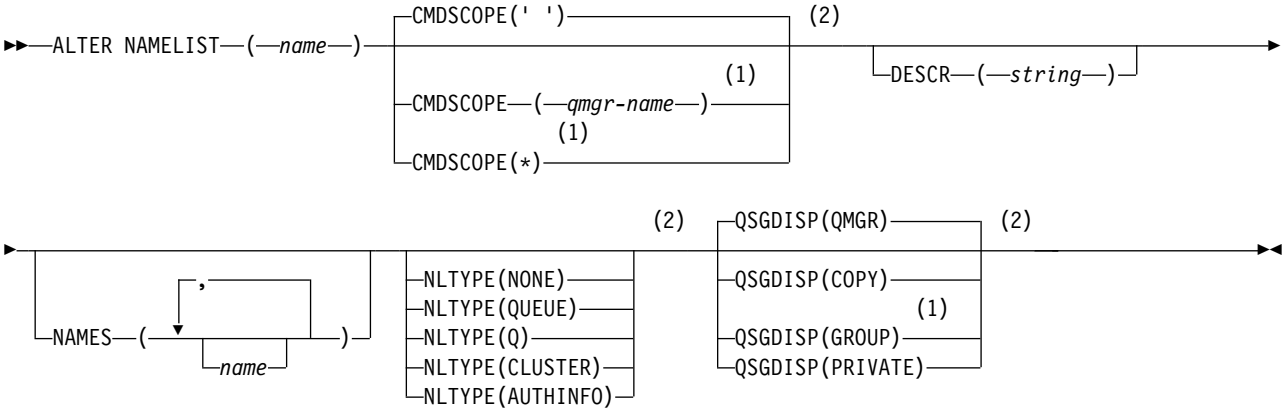
UNIX and Linux	Windows
✓	✓

Parameters not specified in the ALTER NAMELIST command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Usage notes" on page 348
- "Parameter descriptions for ALTER NAMELIST" on page 348

Synonym: ALT NL

ALTER NAMELIST



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

On UNIX systems, the command is valid only on AIX, HP-UX, and Solaris.

Parameter descriptions for ALTER NAMELIST

(name) Name of the list.

The name must not be the same as any other namelist name currently defined on this queue manager (unless REPLACE or ALTER is specified). See Rules for naming IBM WebSphere MQ objects.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of specifying * is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(*string***)**

Plain-text comment. It provides descriptive information about the namelist when an operator issues the DISPLAY NAMELIST command (see "DISPLAY NAMELIST" on page 661).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

NAMES(*name, ...***)**

List of names.

The names can be of any type, but must conform to the rules for naming WebSphere MQ objects, with a maximum length of 48 characters.

An empty list is valid: specify NAMES(). The maximum number of names in the list is 256.

NLTYPE

Indicates the type of names in the namelist.

This parameter is valid only on z/OS.

NONE
The names are of no particular type.

QUEUE or Q
A namelist that holds a list of queue names.

CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

AUTHINFO
This namelist is associated with SSL and contains a list of authentication information object names.

Namelists used for clustering must have NLTYPE(CLUSTER) or NLTYPE(NONE).

Namelists used for SSL must have NLTYPE(AUTHINFO).

QSGDISP
This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero:
	`DEFINE NAMELIST(name)`
	`REPLACE QSGDISP(COPY)`
	
	The ALTER for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with QSGDISP(QMGR) or QSGDISP(COPY). Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

ALTER PROCESS:

Use the MQSC command ALTER PROCESS to alter the parameters of an existing WebSphere MQ process definition.

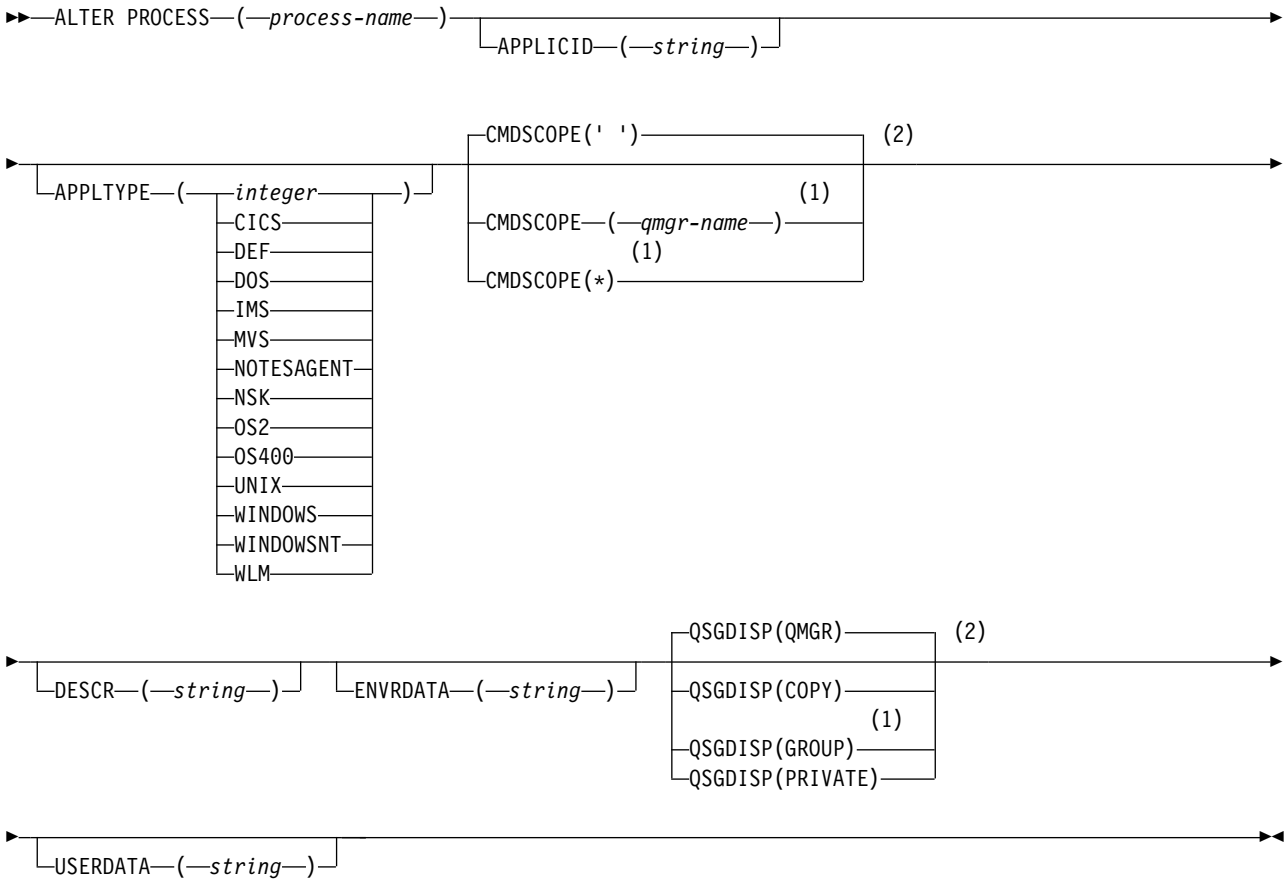
UNIX and Linux	Windows
✓	✓

Parameters not specified in the ALTER PROCESS command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Parameter descriptions for ALTER PROCESS" on page 350

Synonym: `ALT PRO`

ALTER PROCESS



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for ALTER PROCESS

(process-name)
Name of the WebSphere MQ process definition (see Rules for naming IBM WebSphere MQ objects). *process-name* is required.

The name must not be the same as any other process definition currently defined on this queue manager (unless REPLACE is specified).

APPLICID(*string***)**
The name of the application to be started. The name might typically be a fully qualified file name of an executable object. Qualifying the file name is particularly important if you have multiple IBM WebSphere MQ installations, to ensure the correct version of the application is run. The maximum length is 256 characters.

For a CICS application the name is a CICS transaction ID, and for an IMS™ application it is an IMS transaction ID.

On z/OS, for distributed queuing, it must be "CSQX start".

APPLTYPE(*string***)**
The type of application to be started. Valid application types are:

integer
A system-defined application type in the range zero through 65 535 or a user-defined application type in the range 65 536 through 999 999 999.

For certain values in the system range, a parameter from the following list can be specified instead of a numeric value:

CICS Represents a CICS transaction.

DOS Represents a DOS application.

IMS Represents an IMS transaction.

MVS Represents a z/OS application (batch or TSO).

NOTESAGENT
Represents a Lotus Notes® agent.

NSK Represents an HP Integrity NonStop Server application.

OS400 Represents an IBM i application.

UNIX Represents a UNIX application.

WINDOWS
Represents a Windows application.

WINDOWSNT
Represents a Windows NT, Windows 2000, or Windows XP application.

WLM Represents a z/OS workload manager application.

DEF Specifying `DEF` causes the default application type for the platform at which the command is interpreted to be stored in the process definition. This default cannot be changed by the installation. If the platform supports clients, the default is interpreted as the default application type of the server.

Only use application types (other than user-defined types) that are supported on the platform at which the command is executed:

- On z/OS, CICS, DOS, IMS, MVS, OS2, UNIX, WINDOWS, WINDOWSNT, WLM, and DEF are supported
- On IBM i, OS400, CICS, and DEF are supported
- On UNIX systems, UNIX, OS2, DOS, WINDOWS, CICS, and DEF are supported
- On Windows, WINDOWSNT, DOS, WINDOWS, OS2, UNIX, CICS, and DEF are supported

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

`' '` The command is executed on the queue manager on which it was entered.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

In a shared queue environment, you can provide a different queue manager name from the one you are using to enter the command. The command server must be enabled.

`*` The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY PROCESS command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

ENVRDATA(*string***)**
A character string that contains environment information pertaining to the application to be started. The maximum length is 128 characters.

The meaning of ENVRDATA is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ appends ENVRDATA to the parameter list passed to the started application. The parameter list consists of the MQTMC2 structure, followed by one blank, followed by ENVRDATA with trailing blanks removed.

Note:

1. On z/OS, ENVRDATA is not used by the trigger-monitor applications provided by WebSphere MQ.
2. On z/OS, if APPLTYPE is WLM, the default values for the ServiceName and ServiceStep fields in the work information header (MQWIH) can be supplied in ENVRDATA. The format must be:

   ```
   SERVICENAME=servname,SERVICESTEP=stepname
   ```

   where:

   SERVICENAME=
   is the first 12 characters of ENVRDATA.

   servname
   is a 32-character service name. It can contain embedded blanks or any other data, and have trailing blanks. It is copied to the MQWIH as is.

   SERVICESTEP=
   is the next 13 characters of ENVRDATA.

   stepname
   is a 1 - 8 character service step name. It is copied as-is to the MQWIH, and padded to eight characters with blanks.

   If the format is incorrect, the fields in the MQWIH are set to blanks.
3. On UNIX systems, ENVRDATA can be set to the ampersand character to make the started application run in the background.

QSGDISP
This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). On the page set of the queue manager that executes the command, only a local copy of the object is altered by this command. If the command is successful, the following command is generated.
	`DEFINE PROCESS(process-name)`
	`REPLACE QSGDISP(COPY)`
	
	The command is sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero. The ALTER for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with QSGDISP(QMGR) or QSGDISP(COPY). Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

USERDATA(*string***)**

A character string that contains user information pertaining to the application defined in the APPLICID that is to be started. The maximum length is 128 characters.

The meaning of USERDATA is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ simply passes USERDATA to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing USERDATA), followed by one blank, followed by ENVRDATA with trailing blanks removed.

For WebSphere MQ message channel agents, the format of this field is a channel name of up to 20 characters. See Managing objects for triggering for information about what APPLICID to provide to message channel agents.

For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

ALTER QMGR:

Use the MQSC command **ALTER QMGR** to alter the queue manager parameters for the local queue manager.

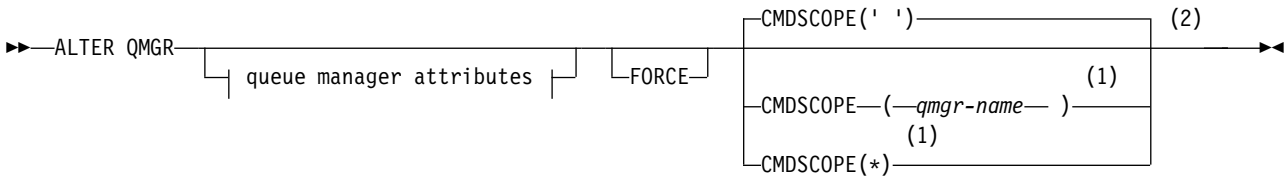
UNIX and Linux	Windows
✓	✓

Parameters not specified in the **ALTER QMGR** command result in the existing values for those parameters being left unchanged.This information is divided into three sections:

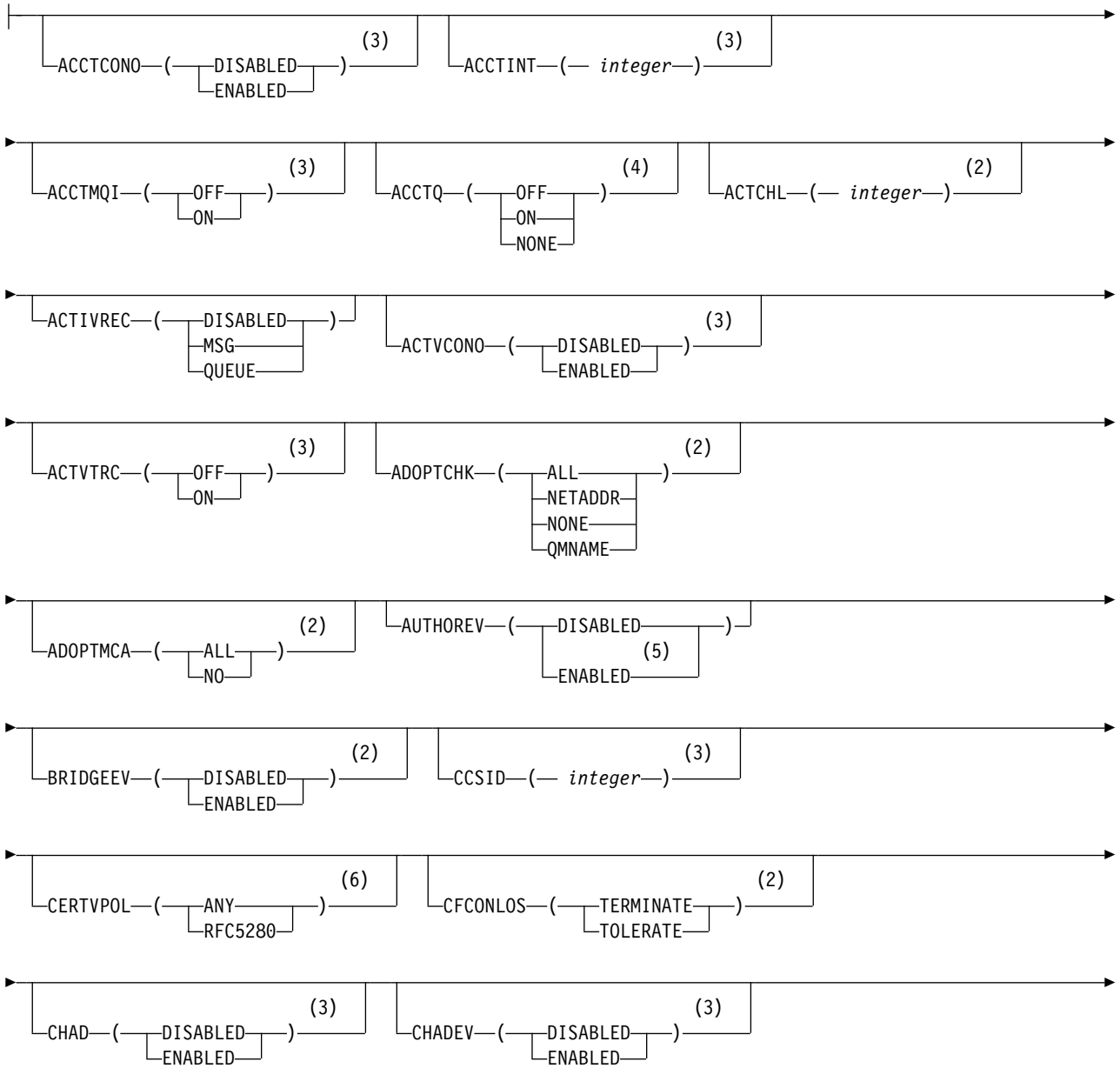
- "ALTER QMGR" on page 354
- "Parameter descriptions for ALTER QMGR" on page 358
- "Queue manager parameters" on page 358

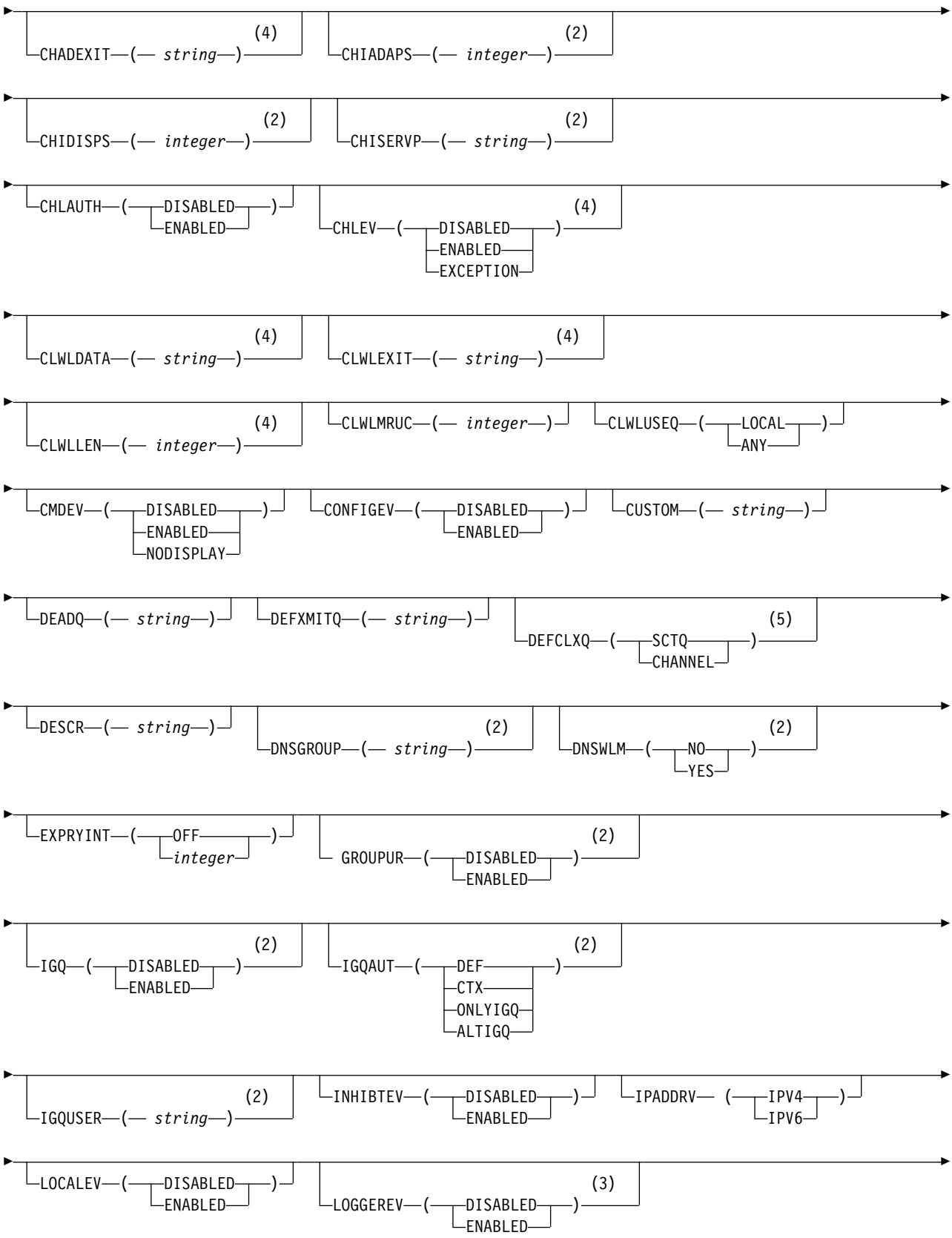
ALTER QMGR

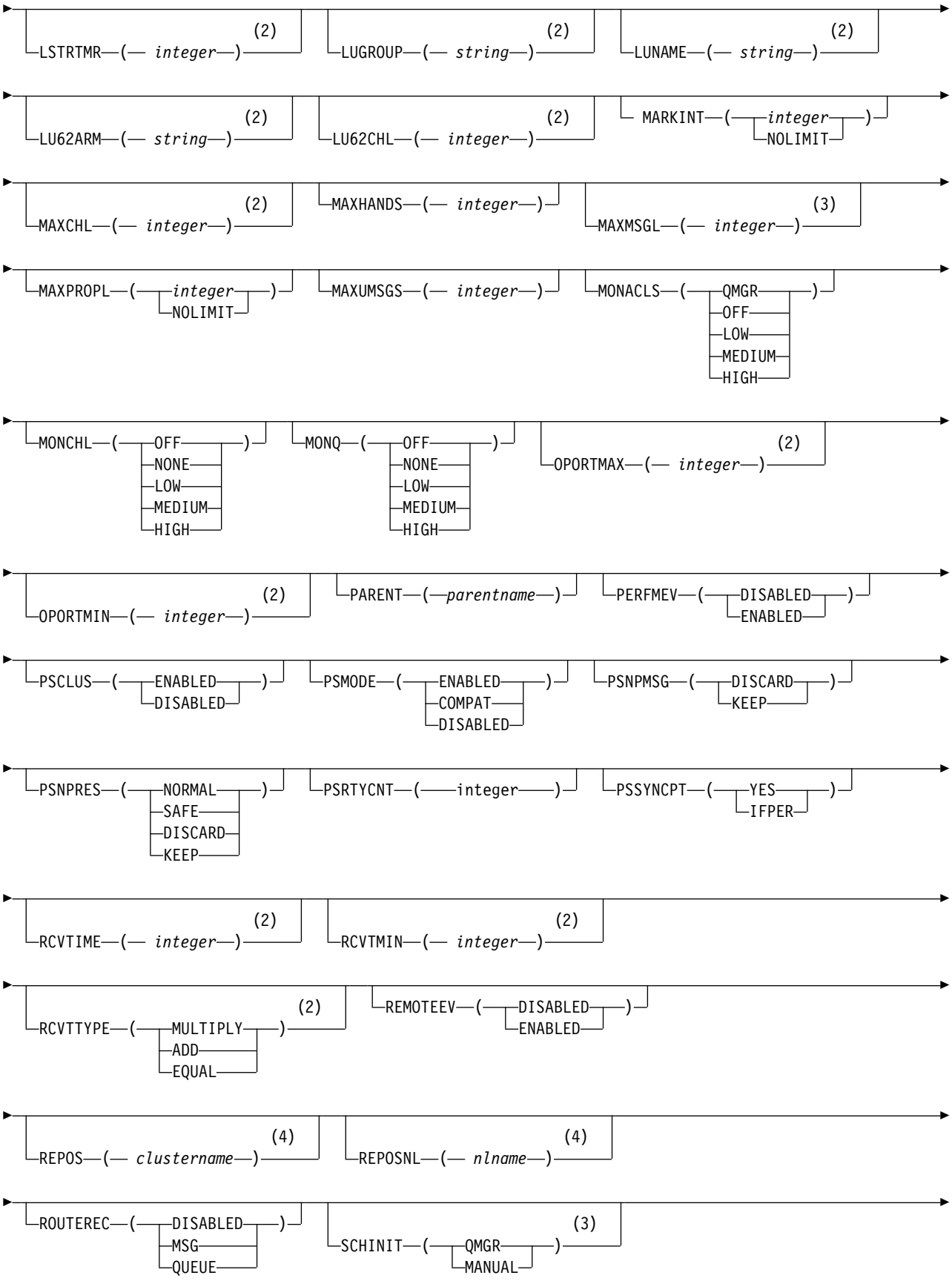
Synonym: ALT QMGR

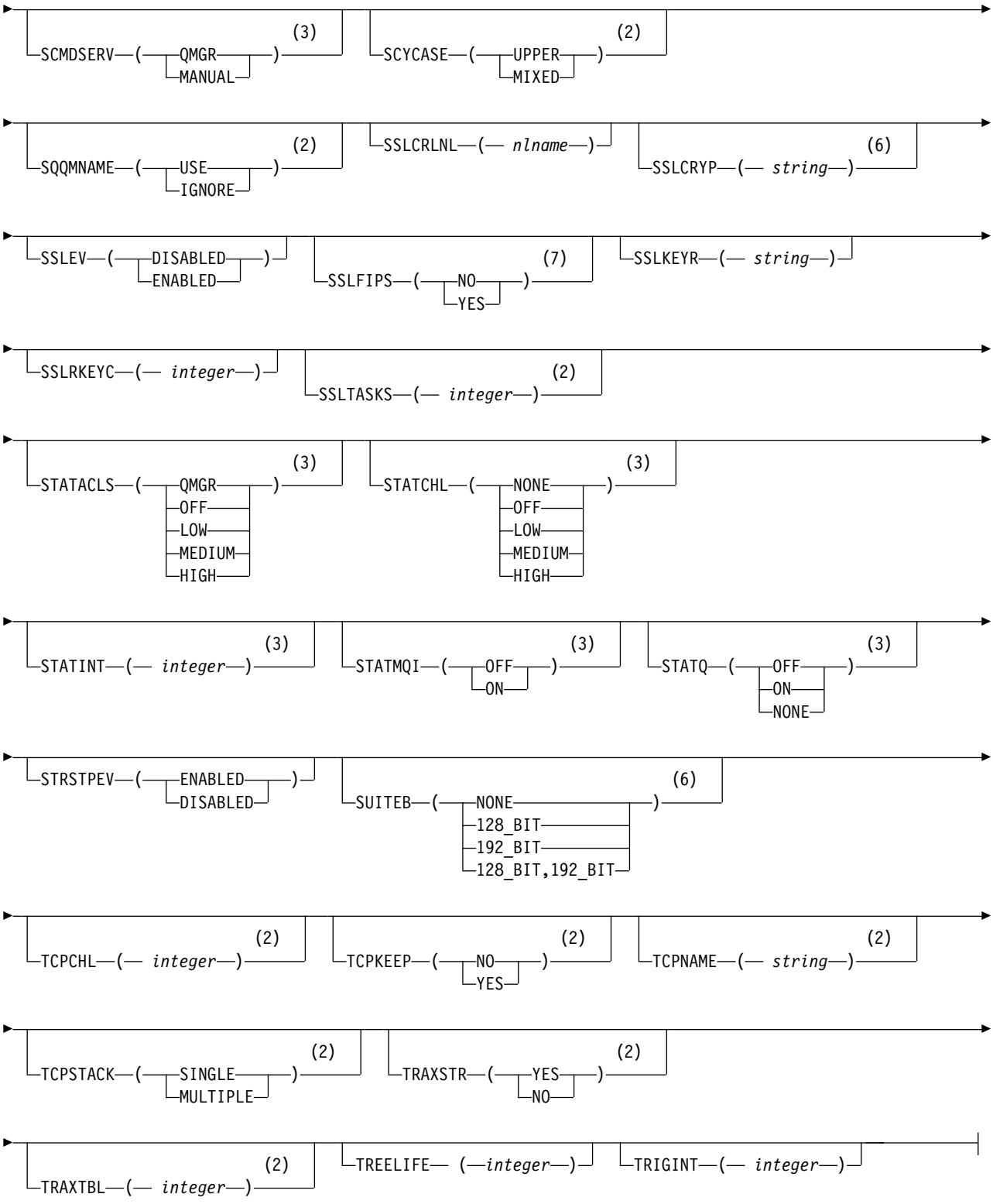


Queue manager attributes:




CHADEXIT—(— string —)(4)
CHIADAPS—(— integer —)(2)
CHIDISPS—(— integer —)(2)
CHISERVP—(— string —)(2)
CHLAUTH—(DISABLED, ENABLED)
CHLEV—(DISABLED, ENABLED, EXCEPTION)(4)
CLWLDATA—(— string —)(4)
CLWLEXIT—(— string —)(4)
CLWLLEN—(— integer —)(4)
CLWLMRUC—(— integer —)
CLWLUSEQ—(LOCAL, ANY)
CMDEV—(DISABLED, ENABLED, NODISPLAY)
CONFIGEV—(DISABLED, ENABLED)
CUSTOM—(— string —)
DEADQ—(— string —)
DEFXMITQ—(— string —)
DEFCLXQ—(SCTQ, CHANNEL)(5)
DESCR—(— string —)
DNSGROUP—(— string —)(2)
DNSWLM—(NO, YES)(2)
EXPRYINT—(OFF, integer)
GROUPUR—(DISABLED, ENABLED)(2)
IGQ—(DISABLED, ENABLED)(2)
IGQAUT—(DEF, CTX, ONLYIGQ, ALTIGQ)(2)
IGQUSER—(— string —)(2)
INHIBTEV—(DISABLED, ENABLED)
IPADDRV—(IPV4, IPV6)
LOCALEV—(DISABLED, ENABLED)
LOGGEREV—(DISABLED, ENABLED)(3)


LSTRTMR (integer) (2)
LUGROUP (string) (2)
LUNAME (string) (2)
LU62ARM (string) (2)
LU62CHL (integer) (2)
MARKINT (integer | NOLIMIT)
MAXCHL (integer) (2)
MAXHANDS (integer)
MAXMSGL (integer) (3)
MAXPROPL (integer | NOLIMIT)
MAXUMSGS (integer)
MONACLS (QMGR | OFF | LOW | MEDIUM | HIGH)
MONCHL (OFF | NONE | LOW | MEDIUM | HIGH)
MONQ (OFF | NONE | LOW | MEDIUM | HIGH)
OPORTMAX (integer) (2)
OPORTMIN (integer) (2)
PARENT (parentname)
PERFMEV (DISABLED | ENABLED)
PSCLUS (ENABLED | DISABLED)
PSMODE (ENABLED | COMPAT | DISABLED)
PSNPMSG (DISCARD | KEEP)
PSNPRES (NORMAL | SAFE | DISCARD | KEEP)
PSRTYCNT (integer)
PSSYNCPT (YES | IFPER)
RCVTIME (integer) (2)
RCVTMIN (integer) (2)
RCVTTYPE (MULTIPLY | ADD | EQUAL) (2)
REMOTEEV (DISABLED | ENABLED)
REPOS (clustername) (4)
REPOSNL (nlname) (4)
ROUTEREC (DISABLED | MSG | QUEUE)
SCHINIT (QMGR | MANUAL) (3)



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Valid only on IBM i, UNIX, Linux, and Windows.

4 Valid only on z/OS, UNIX, Linux, and Windows.

5 Not valid on z/OS.

6 Valid only on UNIX, Linux, and Windows.

7 Not valid on IBM i

Parameter descriptions for ALTER QMGR

The parameters you specify override the current values. Attributes that you do not specify are unchanged.

Note:

1. If you do not specify any parameters, the command completes successfully, but no queue manager options are changed.
2. Changes made using this command persist when the queue manager is stopped and restarted.

FORCE

Specify this parameter to force completion of the command if both of the following are true:

- The DEFXMITQ parameter is specified
- An application has a remote queue open, the resolution for which would be affected by this change

If FORCE is not specified in these circumstances, the command is unsuccessful.

Queue manager parameters

These parameters are the queue manager parameters for the **ALTER QMGR** command:

ACCTCONO

Specifies whether applications can override the settings of the ACCTQ and ACCTMQI queue manager parameters:

DISABLED

Applications cannot override the settings of the ACCTQ and ACCTMQI parameters.

This is the queue manager's initial default value.

ENABLED

Applications can override the settings of the ACCTQ and ACCTMQI parameters by using the options field of the MQCNO structure of the MQCONNX API call.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ACCTINT(*integer***)**

The time interval, in seconds, at which intermediate accounting records are written.

Specify a value in the range 1 through 604800.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ACCTMQI

Specifies whether accounting information for MQI data is to be collected:

OFF MQI accounting data collection is disabled.

This is the queue manager's initial default value.

ON MQI accounting data collection is enabled.

If queue manager attribute ACCTCONO is set to ENABLED, the value of this parameter can be overridden using the options field of the MQCNO structure.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ACCTQ

Specifies whether accounting data is to be collected for all queues. On z/OS, the data collected is class 3 accounting data (thread-level and queue-level accounting).

OFF Accounting data collection is disabled for all queues which specify QMGR as the value for their ACCTQ parameter.

ON Accounting data collection is enabled for all queues which specify QMGR as the value of their ACCTQ parameter. On z/OS systems, you must switch on class 3 accounting by the START TRACE command.

NONE

Accounting data collection for all queues is disabled regardless of the value of the ACCTQ parameter of the queue.

Changes to this parameter are effective only for connections to the queue manager occurring after the change to the parameter.

ACTCHL(*integer* **)**

The maximum number of channels that can be *active* at any time, unless the value is reduced below the number of currently active channels.

Specify a value from 1 through 9999 that is not greater than the value of MAXCHL. MAXCHL defines the maximum number of channels available.

If you change this value, you must also review the MAXCHL, LU62CHL, and TCPCHL values to ensure that there is no conflict of values

For an explanation of which channel states are considered active; see Channel states.

If the value of ACTCHL is reduced to less than its value when the channel initiator was initialized, channels continue to run until they stop. When the number of running channels falls below the value of ACTCHL , more channels can be started. Increasing the value of ACTCHL to more than its value when the channel initiator was initialized does not have immediate effect. The higher value of ACTCHL takes effect at the next channel initiator restart.

Sharing conversations do not contribute to the total for this parameter.

This parameter is valid on z/OS only.

ACTIVREC

Specifies whether activity reports are generated if requested in the message:

DISABLED

Activity reports are not generated.

MSG Activity reports are generated and sent to the reply queue specified by the originator in the message causing the report.

This is the queue manager's initial default value.

QUEUE

Activity reports are generated and sent to SYSTEM.ADMIN.ACTIVITY.QUEUE

See Activity recording.

ACTVCONO
Specifies whether applications can override the settings of the ACTVTRC queue manager parameter:

DISABLED
Applications cannot override the settings of the ACTVTRC queue manager parameter.

This is the queue manager's initial default value.

ENABLED
Applications can override the settings of the ACTVTRC queue manager parameter by using the options field of the MQCNO structure of the MQCONNX API call.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ACTVTRC
Specifies whether MQI application activity tracing information is to be collected. See Setting ACTVTRC to control collection of activity trace information.

OFF WebSphere MQ MQI application activity tracing information collection is not enabled.

This is the queue manager's initial default value.

ON WebSphere MQ MQI application activity tracing information collection is enabled.

If the queue manager attribute ACTVCONO is set to ENABLED, the value of this parameter can be overridden using the options field of the MQCNO structure.

Changes to this parameter are effective for connections to the queue manager that occur after the change.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ADOPTCHK
Specifies which elements are checked to determine whether an MCA is adopted. The check is made when a new inbound channel is detected with the same name as an already active MCA.

ALL Check the queue manager name and the network address. Perform this check to prevent your channels from being inadvertently or maliciously shut down.

This is the queue manager's initial default value.

NETADDR
Check the network address.

NONE
Do no checking.

QMNAME
Check the queue manager name.

This parameter is valid on z/OS only.

Changes to this parameter take effect the next time that a channel attempts to adopt an MCA.

ADOPTMCA
Specifies whether an orphaned instance of an MCA restarts immediately when a new inbound channel request matching the ADOPTCHK parameter is detected:

ALL Adopt all channel types.

This is the queue manager's initial default value.

NO Adoption of orphaned channels is not required.

This parameter is valid on z/OS only

Changes to this parameter take effect the next time that a channel attempts to adopt an MCA.

AUTHOREV

Specifies whether authorization (Not Authorized) events are generated:

DISABLED

Authorization events are not generated.

This is the queue manager's initial default value.

ENABLED

Authorization events are generated.

This value is not supported on z/OS.

BRIDGEEV

Specifies whether IMS Bridge events are generated.

DISABLED

IMS Bridge events are not generated.

This is the queue manager's initial default value.

ENABLED

All IMS Bridge events are generated.

This parameter is valid on z/OS only.

CCSID(*integer* **)**

The coded character set identifier for the queue manager. The CCSID is the identifier used with all character string fields defined by the API. If the CCSID in the message descriptor is set to the value `MQCCSI_Q_MGR` , the value applies to application data in the body of a message. The value is set when the message is put to a queue.

Specify a value in the range 1 through 65535. The CCSID specifies a value that is defined for use on your platform, and use a character set that is appropriate to the platform.

If you use this parameter to change the CCSID, applications that are running when the change is applied continue to use the original CCSID. Therefore, stop and restart all running applications before you continue including the command server and channel programs. To stop and restart all running applications, stop and restart the queue manager after changing the parameter value.

This parameter is not valid on z/OS. See Code page conversion for details of the supported CCSIDs for each platform.

CERTVPOL

Specifies which SSL/TLS certificate validation policy is used to validate digital certificates received from remote partner systems. This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards.

ANY Apply each of the certificate validation policies supported by the secure sockets library and accept the certificate chain if any of the policies considers the certificate chain valid. This setting can be used for maximum backwards compatibility with older digital certificates which do not comply with the modern certificate standards.

RFC5280

Apply only the RFC 5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

For more information about certificate validation policies, see Certificate validation policies in WebSphere MQ.

This parameter is valid on only UNIX, Linux, and Windows. Changes to the parameter take effect only after a **`REFRESH SECURITY TYPE(SSL)`** command is issued.

CFCONLOS
Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with CFCONLOS set to ASQMGR

TERMINATE
The queue manager terminates when connectivity to CF structures is lost.

TOLERATE
The queue manager tolerates loss of connectivity to CF structures without terminating.

This parameter is valid on z/OS only.

All queue managers in the queue-sharing group must be at command level 710 or greater and OPMODE set to NEWFUNC for **TOLERATE** to be selected.

CHAD
Specifies whether receiver and server-connection channels can be defined automatically:

DISABLED
Auto-definition is not used.

This is the queue manager's initial default value.

ENABLED
Auto-definition is used.

Cluster-sender channels can always be defined automatically, regardless of the setting of this parameter.

This parameter is not valid on z/OS.

CHADEV
Specifies whether channel auto-definition events are generated.

DISABLED
Auto-definition events are not generated.

This is the queue manager's initial default value.

ENABLED
Auto-definition events are generated.

This parameter is not valid on z/OS.

CHADEXIT(*string* **)**
Auto-definition exit name.

If this name is nonblank, the exit is called when an inbound request for an undefined receiver, server-connection, or cluster-sender channel is received. It is also called when starting a cluster-receiver channel.

The format and maximum length of the name depends on the environment:

- On Windows, it is of the form *dllname(functionname)* where *dllname* is specified without the suffix .DLL. The maximum length is 128 characters.
- On IBM i, it is of the form:

  ```
  progname libname
  ```

  where *program name* occupies the first 10 characters and *libname* the second 10 characters (both blank-padded to the right if necessary). The maximum length of the string is 20 characters.
- On UNIX, and Linux, it is of the form *libraryname(functionname)*. The maximum length is 128 characters.
- On z/OS, it is a load module name, the maximum length is eight characters.

On z/OS, this parameter applies only to cluster-sender and cluster-receiver channels.

CHIADAPS(*integer* **)**
The number of channel initiator adapter subtasks to use for processing IBM WebSphere MQ calls.

Specify a value in the range 0 - 9999.

Suggested settings:

- Test system: 8
- Production system: 30

This parameter is valid on z/OS only.

Changes to this parameter take effect when the channel initiator is restarted.

CHIDISPS(*integer* **)**
The number of dispatchers to use in the channel initiator.

Specify a value in the range 1 through 9999.

Suggested settings:

- Test system: 5
- Production system: 20

This parameter is valid on z/OS only.

Changes to this parameter take effect when the channel initiator is restarted.

CHISERVP
This parameter is reserved for IBM use only; it is not for general use.

This parameter is valid on z/OS only.

CHLAUTH
Specifies whether the rules defined by channel authentication records are used. `CHLAUTH` rules can still be set and displayed regardless of the value of this attribute.

Changes to this parameter take effect the next time that an inbound channel attempts to start.

Channels that are currently started are unaffected by changes to this parameter.

DISABLED
Channel authentication records are not checked.

ENABLED
Channel authentication records are checked.

CHLEV
Specifies whether channel events are generated.

DISABLED
Channel events are not generated.

This is the queue manager's initial default value.

ENABLED
All channel events are generated.

EXCEPTION
All exception channel events are generated.

CLWLDATA(*string* **)**
Cluster workload exit data. The maximum length of the string is 32 characters.

This string is passed to the cluster workload exit when it is called.

CLWLEXIT(*string* **)**
Cluster workload exit name.

If this name is nonblank, the exit is called when a message is put to a cluster queue. The format and maximum length of the name depends on the environment:

- On UNIX and Linux systems, it is of the form *libraryname(functionname)* . The maximum length is 128 characters.
- On Windows, it is of the form *dllname(functionname)*, where *dllname* is specified without the suffix `.DLL`. The maximum length is 128 characters.
- On z/OS, it is a load module name. The maximum length is eight characters.
- On IBM i, it is of the form:

```
progname libname
```

  where *program name* occupies the first 10 characters and *libname* the second 10 characters (both blank-padded to the right if necessary). The maximum length is 20 characters.

This parameter is valid only on IBM i, z/OS, UNIX, Linux, and Windows.

CLWLLEN(*integer* **)**
The maximum number of bytes of message data that is passed to the cluster workload exit.

Specify a value:

- In the range 0 - 100 MB on IBM WebSphere MQ for z/OS systems
- In the range 0 - 999,999,999 on other platforms

This parameter is valid only on IBM i, z/OS, UNIX, Linux, and Windows.

CLWLMRUC(*integer***)**
The maximum number of most recently used outbound cluster channels.

Specify a value in the range 1 through 999,999,999.

See `CLWLMRUC` queue manager attribute.

CLWLUSEQ
The attribute applies to queues with the queue attribute `CLWLUSEQ` set to `QMGR`. It specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance. It does not apply if the MQPUT originates from a cluster channel.

Specify either:

LOCAL
The local queue is the only target for MQPUT operations.

This is the queue manager's initial default value.

ANY The queue manager treats the local queue as another instance of the cluster queue for the purposes of workload distribution.

See `CLWLUSEQ` queue manager attribute.

CMDEV
Specifies whether command events are generated:

DISABLED
Command events are not generated.

This is the queue manager's initial default value.

ENABLED
Command events are generated for all successful commands.

NODISPLAY
Command events are generated for all successful commands, other than DISPLAY commands.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is run when the queue manager is a member of a queue-sharing group.

' The command is run on the queue manager on which it was entered.

qmgr-name
The command is run on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a different queue manager. You can do so if you are using a queue-sharing group environment, and if the command server is enabled. You can then specify a different queue manager to the one on which the command is entered.

*****

The command is run on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of entering this value is the same as entering the command on every queue manager in the queue-sharing group.

CONFIGEV
Specifies whether configuration events are generated:

ENABLED
Configuration events are generated. After setting this value, issue `REFRESH QMGR TYPE(CONFIGEV)` commands for all objects to bring the queue manager configuration up to date.

DISABLED
Configuration events are not generated.

This is the queue manager's initial default value.

CUSTOM(*string***)**
The custom attribute for new features.

This attribute is reserved for the configuration of new features before named attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form `NAME(VALUE)` . Escape a single quotation mark with another single quotation mark.

No values are defined for *`Custom`* .

DEADQ(*string* **)**
The local name of a dead-letter queue (or undelivered-message queue) on which messages that cannot be routed to their correct destination are put.

The queue named must be a local queue; see Rules for naming IBM WebSphere MQ objects .

DEFCLXQ
The `DEFCLXQ` attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

SCTQ

All cluster-sender channels send messages from `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. The `correlID` of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

`SCTQ` is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute `DEFCLXQ` was not present.

CHANNEL
Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

If the queue manager attribute, DEFCLXQ, is set to CHANNEL, the default configuration is changed to cluster-sender channels being associated with individual cluster transmission queues. The transmission queues are permanent-dynamic queues created from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Each transmission queue is associated with one cluster-sender channel. As one cluster-sender channel services a cluster transmission queue, the transmission queue contains messages for only one queue manager in one cluster. You can configure clusters so that each queue manager in a cluster contains only one cluster queue. In this case, the message traffic from a queue manager to each cluster queue is transferred separately from messages to other queues.

DEFXMITQ(*string* **)**
Local name of the default transmission queue on which messages destined for a remote queue manager are put. The default transmission queue is used if there is no other suitable transmission queue defined.

The cluster transmission queue must not be used as the default transmission queue of the queue manager.

The queue named must be a local transmission queue; see Rules for naming IBM WebSphere MQ objects .

DESCR(*string* **)**
Plain-text comment. It provides descriptive information about the queue manager.

It contains only displayable characters. The maximum length of the string is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

If the characters in the descriptive information are in the coded character set identifier (CCSID) for this queue manager they are translated correctly. They are translated when the descriptive information is sent to another queue manager. If they are not in the CCSID for this queue manager, they might be translated incorrectly.

DNSGROUP(*string* **)**
DNSGROUP applies if you are using Workload Manager for Dynamic Domain Name Services support (WLM/DNS). DNSGROUP is the name of the group that the TCP listener handling inbound transmissions for the queue-sharing group joins when using WLM/DNS.

The maximum length of this parameter is 18 characters.

If this name is blank, the queue-sharing group name is used.

This parameter is valid on z/OS only.

Changes to this parameter take effect for listeners that are later started. Listeners that are currently started are unaffected by changes to this parameter.

DNSWLM
Specifies whether the TCP listener that handles inbound transmissions for the queue-sharing group registers with WLM/DNS:

NO The listener is not to register with Workload Manager.

This is the queue manager's initial default value.

YES The listener is to register with Workload Manager.

This parameter is valid on z/OS only.

Changes to this parameter take effect for listeners that are later started. Listeners that are currently started are unaffected by changes to this parameter.

EXPRYINT

Specifies how often queues are scanned to discard expired messages:

OFF Queues are not scanned. No internal expiry processing is performed.

integer The approximate interval in seconds at which queues are scanned. Each time that the expiry interval is reached, the queue manager looks for candidate queues that are worth scanning to discard expired messages.

The queue manager maintains information about the expired messages on each queue, and therefore whether a scan for expired messages is worthwhile. So, only a selection of queues is scanned at any time.

The value must be in the range 1 through 99999999. The minimum scan interval used is 5 seconds, even if you specify a lower value.

You must set the same `EXPRYINT` value for all queue managers within a queue-sharing group that support this attribute. Shared queues are scanned by only one queue manager in a queue-sharing group. This queue manager is either the first queue manager to restart, or the first queue manager for which `EXPRYINT` is set.

Changes to `EXPRYINT` take effect when the current interval expires. Changes also take effect if the new interval is less than the unexpired portion of the current interval. In this case, a scan is scheduled and the new interval value takes immediate effect.

This parameter is valid only on z/OS.

GROUPUR

This parameter controls whether CICS and XA client applications can establish transactions with a GROUP unit of recovery disposition.

This parameter is valid only on z/OS. The property can be enabled only when the queue manager is a member of a queue-sharing group.

ENABLED
CICS and XA client applications can establish transactions with a group unit of recovery disposition by specifying a queue-sharing group name when they connect.

DISABLED
CICS and XA client applications must connect using a queue manager name.

IGQ Specifies whether intra-group queuing is used.

This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

ENABLED
Message transfer between queue managers within a queue-sharing group uses the shared transmission queue, `SYSTEM.QSG.TRANSMIT.QUEUE` .

DISABLED
Message transfer between queue managers within a queue-sharing group uses non-shared transmission queues and channels. Queue managers that are not part of a queue-sharing group also use this mechanism.

If intra-group queuing is enabled, but the intra-group queuing agent is stopped, issue `ALTER QMGR IGQ(ENABLED)` to restart it.

IGQAUT

Specifies the type of authority checking and, therefore, the user IDs, to be used by the `IGQ` agent (IGQA). This parameter establishes the authority to put messages to a destination queue.

This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

DEF Indicates that the default user ID is used to establish authority to put messages to a destination queue.

For a one user ID check, the default user ID is the user ID of a queue manager within the queue-sharing group. The default user ID is the user ID of the queue manager that put the messages to the SYSTEM.QSG.TRANSMIT.QUEUE . This user ID is referred to as the QSGSEND user ID.

For two user ID checks, the default second user ID is the IGQ user ID.

CTX Indicates that the user ID from a *UserIdentifier* field is used to establish authority to put messages to a destination queue. The user ID is the *UserIdentifier* field in the message descriptor of a message on the SYSTEM.QSG.TRANSMIT.QUEUE.

For one user ID check, the QSGSEND user ID is used.

For two user ID checks, the QSGSEND user ID, the IGQ user ID and the alternate user ID are used. The alternate user ID is taken from the *UserIdentifier* field in the message descriptor of a message on the SYSTEM.QSG.TRANSMIT.QUEUE. The alternate user ID is referred to as ALT.

ONLYIGQ
Indicates that only the IGQ user ID is used to establish authority to put messages to a destination queue.

For all ID checks, the IGQ user ID is used.

ALTIGQ
Indicates that the IGQ user ID and the ALT user ID are used to establish authority to put messages to a destination queue.

For one user ID check, the IGQ user ID is used.

For two user ID checks, the IGQ user ID and the ALT user ID are used.

IGQUSER
Nominates a user ID to be used by the IGQ agent (IGQA) to establish authority to put messages to a destination queue. The user ID is referred to as the IGQ user ID.

This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group. Possible values are:

Blanks
Indicates that the user ID of the receiving queue manager within the queue-sharing group is used.

Specific user ID
Indicates that the user ID specified in the IGQUSER parameter of the receiving queue manager is used.

Note:

1. As the receiving queue manager has authority to all queues it can access, security checking might not be performed for this user ID type.
2. As the value of blanks has a special meaning, you cannot use IGQUSER to specify a real user ID of blanks.

INHIBTEV
Specifies whether inhibit events are generated. The events are generated for Inhibit Get and Inhibit Put)

ENABLED
Inhibit events are generated.

DISABLED
Inhibit events are not generated.

This is the queue manager's initial default value.

IPADDRV
Specifies which IP protocol is to be used for channel connections.

IPV4 The IPv4 IP address is to be used.

This is the queue manager's initial default value.

IPV6 The IPv6 IP address is to be used.

This parameter is used only in systems running IPv4 and IPv6. It applies to channels defined only with a TRPTYPE of TCP when either of the following two conditions is true:

- The CONNAME parameter of the channel contains a host name that resolves to both an IPv4 and an IPv6 address, and the LOCLADDR parameter is not specified.
- The value of the CONNAME and LOCLADDR parameters of the channel is a host name that resolves to both an IPv4 and IPv6 address.

LOCALEV
Specifies whether local error events are generated:

ENABLED
Local error events are generated.

DISABLED
Local error events are not generated.

This is the queue manager's initial default value.

LOGGEREV
Specifies whether recovery log events are generated:

DISABLED
Logger events are not generated.

This is the queue manager's initial default value.

ENABLED
Logger events are generated.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

LSTRTMR(*integer* **)**
The time interval, in seconds, between attempts by IBM WebSphere MQ to restart a listener after an APPC or TCP/IP failure. When the listener is restarted on TCP/IP, it uses the same port and IP address as it used when it first started.

Specify a value in the range 5 through 9999.

This parameter is valid on z/OS only.

Changes to this parameter take effect for listeners that are later started. Listeners that are currently started are unaffected by changes to this parameter.

LUGROUP(*string* **)**
The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group. The maximum length of this parameter is eight characters.

If this name is blank, the listener cannot be used.

This parameter is valid on z/OS only.

Changes to this parameter take effect for listeners that are later started. Listeners that are currently started are unaffected by changes to this parameter.

LUNAME(*string* **)**
The name of the LU to use for outbound LU 6.2 transmissions. Set this parameter to be the same as the name of the LU to be used by the listener for inbound transmissions. The maximum length of this parameter is eight characters.

If this name is blank, the APPC/MVS default LU name is used. This name is variable, so LUNAME must always be set if you are using LU 6.2

This parameter is valid on z/OS only.

Changes to this parameter take effect when the channel initiator is restarted.

LU62ARM(*string* **)**
The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. When automatic restart manager (ARM) restarts the channel initiator, the z/OS command `SET APPC=` *xx* is issued.

If you do not provide a value for this parameter, no `SET APPC=`*xx* command is issued.

The maximum length of this parameter is two characters.

This parameter is valid on z/OS only.

Changes to this parameter take effect when the channel initiator is restarted.

LU62CHL(*integer* **)**
The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

Specify a value 0- 9999 that is not greater than the value of `MAXCHL`. `MAXCHL` defines the maximum number of channels available. If you specify zero, the LU 6.2 transmission protocol is not used.

If you change this value, also review the `MAXCHL`, `LU62CHL`, and `ACTCHL` values. Ensure that there is no conflict of values and if necessary, raise the value of `MAXCHL` and `ACTCHL`.

This parameter is valid on z/OS only.

If the value of this parameter is reduced, any current channels that exceed the new limit continue to run until they stop.

MARKINT(*integer***)**
The time interval, expressed in milliseconds, for which messages marked as browsed by a call to MQGET, with the get message option `MQGMO_MARK_BROWSE_CO_OP`, are expected to remain mark-browsed.

If messages are marked for more than approximately `MARKINT` milliseconds, the queue manager might automatically unmark messages. It might unmark messages that are marked as browsed for the cooperating set of handles.

This parameter does not affect the state of any message marked as browse by a call to MQGET with the get message option `MQGMO_MARK_BROWSE_HANDLE`.

Specify a value in the range 0 - 999,999,999.

The special value `NOLIMIT` indicates that the queue manager does not automatically unmark messages by this process.

MAXCHL(*integer* **)**
The maximum number of channels that can be *current* (including server-connection channels with connected clients).

Specify a value in the range 1- 9999. If you change this value, also review the `TCPCHL`, `LU62CHL` , and `ACTCHL` values to ensure that there is no conflict of values. If necessary, increase the number of active channels with the `ACTCHL` value. The values of `ACTCHL` , `LU62CHL`, and `TCPCHL` must not be greater than the maximum number of channels.

Suggested settings:

- Test system: 200
- Production system: 1000

For an explanation of which channel states are considered current; see Channel states.

If the value of this parameter is reduced, any current channels that exceed the new limit continue to run until they stop.

If the value of MAXCHL is reduced to less than its value when the channel initiator was initialized, channels continue to run until they stop. When the number of running channels falls below the value of MAXCHL, more channels can be started. Increasing the value of MAXCHL to more than its value when the channel initiator was initialized does not have immediate effect. The higher value of MAXCHL takes effect at the next channel initiator restart.

Sharing conversations do not contribute to the total for this parameter.

This parameter is valid on z/OS only.

MAXHANDS(*integer* **)**
The maximum number of open handles that any one connection can have at the same time.

This value is a value in the range 0 - 999,999,999.

MAXMSGL(*integer* **)**
The maximum length of messages allowed on queues for this queue manager.

This value is in the range 32 KB through 100 MB.

Ensure that you also consider the length of any message properties when deciding the value for the MAXMSGL parameter of a channel.

If you reduce the maximum message length for the queue manager, you must also reduce the maximum message length of the SYSTEM.DEFAULT.LOCAL.QUEUE definition. You must also reduce the maximum message length for all other queues connected to the queue manager. This change ensures that the limit of the queue manager is not less than the limit of any of the queues associated with it. If you do not change these lengths, and applications inquire only the MAXMSGL value of the queue, they might not work correctly.

Note that by adding the digital signature and key to the message, WebSphere MQ Advanced Message Security increases the length of the message.

MAXPROPL(*integer***)**
The maximum length of property data in bytes that can be associated with a message.

This value is in the range 0 through 100 MB (104 857 600 bytes).

The special value NOLIMIT indicates that the size of the properties is not restricted, except by the upper limit.

MAXUMSGS(*integer* **)**
The maximum number of uncommitted messages within a sync point.

MAXUMSGS is a limit on the number of messages that can be retrieved, plus the number of messages that can be put, within any single sync point. The limit does not apply to messages that are put or retrieved outside sync point.

The number includes any trigger messages and report messages generated within the same unit of recovery.

If existing applications and queue manager processes are putting and getting a larger number of messages in sync point, reducing MAXUMSGS might cause problems. An example of queue manager processes that might be affected is clustering on z/OS.

Specify a value in the range 1 through 999,999,999. The default value is 10000.

MAXUMSGS has no effect on IBM WebSphere MQ Telemetry. IBM WebSphere MQ Telemetry tries to batch requests to subscribe, unsubscribe, send, and receive messages from multiple clients into batches of work within a transaction.

MONACLS
Controls the collection of online monitoring data for auto-defined cluster-sender channels:

QMGR
Collection of online monitoring data is inherited from the setting of the MONCHL parameter of the queue manager.

This is the queue manager's initial default value.

OFF Monitoring for the channel is switched off.

LOW Unless MONCHL is NONE, monitoring is switched on with a low rate of data collection with a minimal effect on system performance. The data collected is not likely to be the most current.

MEDIUM
Unless MONCHL is NONE, monitoring is switched on with a moderate rate of data collection with limited effect on system performance.

HIGH Unless MONCHL is NONE, monitoring is switched on with a high rate of data collection with a likely effect on system performance. The data collected is the most current available.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

MONCHL
Controls the collection of online monitoring data for channels. The channels defined with MONCHL(QMGR) are affected by changing the QMGR MONCHL attribute.

OFF Online monitoring data collection is turned off for channels specifying a value of QMGR in their MONCHL parameter.

This is the queue manager's initial default value.

NONE
Online monitoring data collection is turned off for channels regardless of the setting of their MONCHL parameter.

LOW Online monitoring data collection is turned on, with a low ratio of data collection, for channels specifying a value of QMGR in their MONCHL parameter.

MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of QMGRin their MONCHL parameter.

HIGH Online monitoring data collection is turned on, with a high ratio of data collection, for channels specifying a value of QMGR in their MONCHL parameter.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

MONQ
Controls the collection of online monitoring data for queues.

OFF Online monitoring data collection is turned off for queues specifying a value of QMGR in their MONQ parameter.

This is the queue manager's initial default value.

NONE
Online monitoring data collection is turned off for queues regardless of the setting of their MONQ parameter.

LOW Online monitoring data collection is turned on for queues specifying a value of QMGR in their MONQ parameter.

MEDIUM
Online monitoring data collection is turned on for queues specifying a value of QMGR in their MONQ parameter.

HIGH Online monitoring data collection is turned on for queues specifying a value of QMGR in their MONQ parameter.

In contrast to MONCHL, there is no distinction between the values LOW, MEDIUM , and HIGH. These values all turn data collection on, but do not affect the rate of collection.

Changes to this parameter are effective only for queues opened after the parameter is changed.

OPORTMAX(*integer* **)**
The maximum value in the range of port numbers to be used when binding outgoing channels. When all the port numbers in the specified range are used, outgoing channels bind to any available port number.

Specify a value in the range 0 - 65535. A value of zero means that all outgoing channels bind to any available port number.

Specify a corresponding value for OPORTMIN to define a range of port numbers. Ensure that the value you specify for OPORTMAX is greater than or equal to the value you specify for OPORTMIN.

This parameter is valid on z/OS only.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

OPORTMIN(*integer* **)**
The minimum value in the range of port numbers to be used when binding outgoing channels. When all the port numbers in the specified range are used, outgoing channels bind to any available port number.

Specify a value in the range 0 - 65535.

Specify a corresponding value for OPORTMAX to define a range of port numbers. Ensure that the value you specify for OPORTMIN is less than or equal to the value you specify for OPORTMAX .

This parameter is valid on z/OS only.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

PARENT(*parentname* **)**
The name of the parent queue manager to which the local queue manager is to connect as its child in a hierarchy.

A blank value indicates that the queue manager has no parent queue manager.

If there is an existing parent queue manager it is disconnected.

IBM WebSphere MQ hierarchical connections require that the queue manager attribute PSMODE is set to ENABLED.

The value of PARENT can be set to a blank value if PSMODE is set to DISABLED.

Before a queue manager can connect to a queue manager as its child in a hierarchy, channels must exist in both directions. The channels must exist between the parent queue manager and the child queue manager.

If a parent is already defined, the ALTER QMGR PARENT command disconnects from the original parent and sends a connection flow to the new parent queue manager.

Successful completion of the command does not mean that the action completed, or that it is going to complete successfully. Use the DIS PUBSUB TYPE(PARENT) ALL command to track the status of the requested parent relationship.

PERFMEV

Specifies whether performance-related events are generated:

ENABLED

Performance-related events are generated.

DISABLED

Performance-related events are not generated.

This is the queue manager's initial default value.

On IBM WebSphere MQ for z/OS, all the queue managers in a queue-sharing group must have the same setting.

PSCLUS

Controls whether this queue manager participates in publish subscribe activity across any clusters in which it is a member. No clustered topic objects can exist in any cluster when modifying from ENABLED to DISABLED.

For more information about **PSCLUS** and inhibiting clusters publish/subscribe, see Inhibiting clustered publish/subscribe in a cluster .

ENABLED

This queue manager can define clustered topic objects, publish to subscribers on other queue managers, and register subscriptions that receive publications from other queue managers. All queue managers in the cluster running a version of IBM WebSphere MQ that supports this option must specify PSCLUS(ENABLED) for the publish/subscribe activity to function as expected. ENABLED is the default value when a queue manager is created.

DISABLED

This queue manager cannot define clustered topic objects and ignores their definition on any other queue manager in the cluster.

Publications are not forwarded to subscribers elsewhere in the cluster, and subscriptions are not registered other than on the local queue manager.

To ensure that no publish/subscribe activity occurs in the cluster, all queue managers must specify PSCLUS(DISABLED). As a minimum, full repositories must be consistent in enabling or disabling publish/subscribe participation.

PSMODE

Controls whether the publish/subscribe engine and the queued publish/subscribe interface are running. It controls whether applications can publish or subscribe by using the application programming interface. It also controls whether the queues that are monitored by the queued publish/subscribe interface, are monitored.

Changing the PSMODE attribute can change the PSMODE status. Use DISPLAY PUBSUB, or on IBM i **DSPMQM**, to determine the current state of the publish/subscribe engine and the queued publish/subscribe interface.

COMPAT

The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface.

The queued publish/subscribe interface is not running. Any publish/subscribe messages put to the queues that are monitored by the queued publish/subscribe interfaces are not acted upon.

Use this setting for compatibility with WebSphere Message Broker V6 or earlier versions that use this queue manager. WebSphere Message Broker must read the same queues from which the queued publish/subscribe interface would normally read.

DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe by using the application programming interface. Any publish/subscribe messages put to the queues that are monitored by the queued publish/subscribe interfaces are not acted upon.

If a queue manager is in a publish/subscribe cluster or hierarchy, it might receive publish/subscribe messages from other queue managers in the cluster or hierarchy. Examples of such messages are publication messages or proxy subscriptions. While PSMODE is set to DISABLED those messages are not processed. For this reason, disable any queue manager in a publish/subscribe cluster or hierarchy only for as long as there is little build-up of messages.

ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface.

This is the queue manager's initial default value.

Note: If a queue manager is in a publish/subscribe cluster or hierarchy, and you change PSMODE to ENABLED, you might have to run the command REFRESH QMGR TYPE(PROXY). The command ensures that non-durable subscriptions are known across the cluster or hierarchy when PSMODE is set back to ENABLED. The circumstance in which you must run the command is as follows. If PSMODE is changed from ENABLED to DISABLED and back to ENABLED, and one or more non-durable subscriptions exist across all three stages.

PSNPMSG
If the queued publish/subscribe interface cannot process a non-persistent input message it might attempt to write the input message to the dead-letter queue. Whether it attempts to do so depends on the report options of the input message. The attempt to write the input message to the dead-letter queue might fail. In this case, the queued publish/subscribe interface might discard the input message. If MQRO_DISCARD_MSG is specified on the input message, the input message is discarded. If MQRO_DISCARD_MSG is not set, setting PSNPMSG to KEEP prevents the input message from being discarded. The default is to discard the input message.

Note: If you specify a value of IFPER for PSSYNCPT, you must not specify a value of KEEP for PSNPMSG .

DISCARD
Non-persistent input messages might be discarded if they cannot be processed.

KEEP Non-persistent input messages are not discarded if they cannot be processed. In this situation, the queued publish/subscribe interface continues to try to process this message again at appropriate intervals and does not continue processing subsequent messages.

PSNPRES
The PSNPRES attribute controls whether the queued publish/subscribe interface writes an undeliverable reply message to the dead-letter queue, or discards the message. The choice is necessary if the queued publish/subscribe interface cannot deliver a reply message to the reply-to queue.

For new queue managers, the initial value is NORMAL. If you specify a value of IFPER for PSSYNCPT, you must not specify a value of KEEP or SAFE for PSNPRES .

For migrated queue managers on IBM i, UNIX, Linux, and Windows systems, the value depends on DLQNonPersistentResponse and DiscardNonPersistentResponse.

NORMAL
Non-persistent responses which cannot be placed on the reply queue are put on the dead-letter queue. If they cannot be placed on the dead-letter queue then they are discarded.

SAFE Non-persistent responses which cannot be placed on the reply queue are put on the dead-letter queue. If the response cannot be sent and cannot be placed on the dead-letter queue, the queued publish/subscribe interface backs out of the current operation. It tries again at appropriate intervals, and does not continue processing subsequent messages.

DISCARD
Non-persistent responses which cannot be placed on the reply queue are discarded

KEEP

Non-persistent responses are not placed on the dead-letter queue or discarded. Instead the queued publish/subscribe interface backs out the current operation and then tries it again at appropriate intervals and does not continue processing subsequent messages.

PSRTYCNT
If the queued publish/subscribe interface fails to process a command message under sync point, the unit of work is backed out. The command tries to process the message a number of times again, before the publish/subscribe broker processes the command message according to its report options instead. This situation can arise for a number of reasons. For example, if a publish message cannot be delivered to a subscriber, and it is not possible to put the publication on the dead letter queue.

The initial value for this parameter on a new queue manager is 5.

Range is 0 - 999,999,999.

PSSYNCPT
Controls whether the queued publish/subscribe interface processes command messages (publishes or delete publication messages) under sync point.

YES All messages are processed under sync point.

IFPER Only persistent messages are part of the sync point

The initial value of the queue manager is IFPER.

RCVTIME(*integer* **)**
The approximate length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. This parameter applies only to message channels and not to MQI channels.

This number can be qualified as follows:

- To specify that this number is a multiplier to apply to the negotiated HBINT value to determine how long a channel is to wait, set RCVTTYPE to MULTIPLY. Specify an RCVTIME value of zero or in the range 2 through 99. If you specify zero, the channel continues to wait indefinitely to receive data from its partner.
- To specify that RCVTIME is the number of seconds to add to the negotiated HBINT value to determine how long a channel is to wait, set RCVTTYPE to ADD. Specify an RCVTIME value in the range 1 through 999999.
- To specify that RCVTIME is a value, in seconds, that the channel is to wait, set RCVTTYPE to EQUAL. Specify an RCVTIME value in the range 0 - 999,999. If you specify zero, the channel continues to wait indefinitely to receive data from its partner.

This parameter is valid on z/OS only.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

RCVTMIN(*integer* **)**
The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state. This parameter applies only to message channels (and not to MQI channels).

The TCP/IP channel wait time is relative to the negotiated value of HBINT. If RCVTYPE is MULTIPLY, the resultant value might less than the RCVTMIN. In this case, the TCP/IP channel wait time is set to RCVTMIN.

Specify a value, in seconds, between zero and 999999.

This parameter is valid on z/OS only.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

RCVTTYPE
The qualifier to apply to the value in RCVTIME .

MULTIPLY
Specifies that RCVTIME is a multiplier to be applied to the negotiated HBINT value to determine how long a channel waits.

ADD Specifies that RCVTIME is a value, in seconds, to be added to the negotiated HBINT value to determine how long a channel waits.

EQUAL
Specifies that RCVTIME is a value, in seconds, representing how long the channel waits.

This parameter is valid on z/OS only.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

REMOTEEV
Specifies whether remote error events are generated:

DISABLED
Remote error events are not generated.

This is the queue manager's initial default value.

ENABLED
Remote error events are generated.

If you are using the reduced function form of IBM WebSphere MQ for z/OS supplied with WebSphere Application Server, only DISABLED is valid.

REPOS(*clustername* **)**
The name of a cluster for which this queue manager provides a repository manager service. The maximum length is 48 characters conforming to the rules for naming IBM WebSphere MQ objects.

No more than one of the resultant values of REPOS and REPOSNL can be nonblank.

If you use the REPOS parameter to create a full repository queue manager, connect it to at least one other full repository queue manager in the cluster. Connect it using a cluster-sender channel. See the information in Components of a cluster for details about using cluster-sender channels with full repository queue managers.

This parameter is valid on IBM i, z/OS, UNIX, Linux, and Windows.

REPOSNL(*nlname* **)**
The name of a namelist of clusters for which this queue manager provides a repository manager service.

No more than one of the resultant values of REPOS and REPOSNL can be nonblank.

Both REPOS and REPOSNL might be blank, or REPOS might be blank and the namelist specified by REPOSNL is empty. In these cases, this queue manager does not have a full repository. It might be a client of other repository services that are defined in the cluster.

If you use the REPOSNL parameter to create a full repository queue manager, connect it to other full repository queue managers. Connect it to at least one other full repository queue manager in each cluster specified in the namelist using cluster-sender channels. See the information in Components of a cluster for details about using cluster-sender channels with full repository queue managers.

This parameter is valid on IBM i, z/OS, UNIX, Linux, and Windows.

ROUTEREC
Specifies whether trace-route information is recorded if requested in the message. If this parameter is not set to DISABLED, it controls whether any reply generated is sent to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, or to the destination specified by the message itself. If ROUTEREC is not DISABLED, messages not yet at the final destination might have information added to them.

DISABLED
Trace-route information is not recorded.

MSG Trace-route information is recorded and sent to the destination specified by the originator of the message causing the trace route record.

This is the queue manager's initial default value.

QUEUE
Trace-route information is recorded and sent to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE .

SCHINIT
Specifies whether the channel initiator starts automatically when the queue manager starts.

QMGR
The channel initiator starts automatically when the queue manager starts.

MANUAL
The channel initiator does not start automatically.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

SCMDSERV
Specifies whether the command server starts automatically when the queue manager starts.

QMGR
The command server starts automatically when the queue manager starts.

MANUAL
The command server does not start automatically.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

SCYCASE
Specifies whether the security profiles are uppercase or mixed case.

UPPER
The security profiles are uppercase only. However, MXTOPIC and GMXTOPIC are used for topic security, and can contain mixed-case profiles.

MIXED
The security profiles are mixed case. MQCMDS and MQCONN are used for command and connection security but they can contain only uppercase profiles.

Changes to SCYCASE become effective after you run the following command:

```
REFFRESH SECURITY(*) TYPE(CLASSES)
```

This parameter is valid only on z/OS

SQQMNAME
The SQQMNAME attribute specifies whether a queue manager in a queue-sharing group opens a shared queue in the same group directly. The processing queue manager calls MQOPEN for a shared queue and sets the *ObjectQmgrName* parameter for the queue. If the shared queue is in the same queue-sharing group as the processing queue manager, the queue can be opened directly by the processing queue manager. Set the SQQMNAME attribute to control if the queue is opened directly, or by the *ObjectQmgrName* queue manager.

USE The *ObjectQmgrName* is used, and the appropriate transmission queue is opened.

IGNORE
The processing queue manager opens the shared queue directly. Setting the parameter to this value can reduce the traffic in your queue manager network.

This parameter is valid only on z/OS.

SSLCRLNL(*nlname***)**

The name of a namelist of authentication information objects which are used to provide certificate revocation locations to allow enhanced TLS/SSL certificate checking.

If SSLCRLNL is blank, certificate revocation checking is not invoked unless one of the SSL certificates used contains an AuthorityInfoAcces s or CrlDistributionPoint X.509 certificate extension.

Changes to SSLCRLNL, or to the names in a previously specified namelist, or to previously referenced authentication information objects become effective either:

- On IBM i, UNIX, Linux, and Windows systems when a new channel process is started.
- For channels that run as threads of the channel initiator on IBM i, UNIX, Linux, and Windows systems, when the channel initiator is restarted.
- For channels that run as threads of the listener on IBM i, UNIX, Linux, and Windows systems, when the listener is restarted.
- On z/OS, when the channel initiator is restarted.
- When a REFRESH SECURITY TYPE(SSL) command is issued.
- On IBM i queue managers, this parameter is ignored. However, it is used to determine which authentication information objects are written to the AMQCLCHL.TAB file.

SSLCRYP(*string***)**
Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

All supported cryptographic hardware supports the PKCS #11 interface. Specify a string of the following format:

```
GSK_PKCS11=<the PKCS #11 driver path and file name>
;<the PKCS #11 token label>;
<the PKCS #11 token password>;<symmetric cipher setting>
;
```

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver file name is the name of the shared library. An example of the value required for the PKCS #11 driver path and file name is /usr/lib/pkcs11/PKCS11_API.so

To access symmetric cipher operations through GSKit, specify the symmetric cipher setting parameter. The value of this parameter is either:

SYMMETRIC_CIPHER_OFF
Do not access symmetric cipher operations.

SYMMETRIC_CIPHER_ON
Access symmetric cipher operations.

If the symmetric cipher setting parameter is not specified, it has the same effect as specifying SYMMETRIC_CIPHER_OF F.

The maximum length of the string is 256 characters.

If you specify a string that is not in the format listed, you get an error.

When the SSLCRYP value is changed, the cryptographic hardware parameters specified become the ones used for new SSL connection environments. The new information becomes effective:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.
- When a REFRESH SECURITY TYPE(SSL) command is issued.

SSLEV

Specifies whether SSL events are generated.

DISABLED
SSL events are not generated.

This is the queue manager's initial default value.

ENABLED
All SSL events are generated.

SSLFIPS

This parameter is valid only on z/OS, UNIX, Linux, and Windows systems.

SSLFIPS specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM WebSphere MQ, rather than in cryptographic hardware. If cryptographic hardware is configured, the cryptographic modules used are those modules provided by the hardware product. These might, or might not, be FIPS-certified to a particular level. Whether the modules are FIPS-certified depends on the hardware product in use. For more information about FIPS, see the Federal Information Processing Standards (FIPS) manual.

NO If you set SSLFIPS to NO, you can use either FIPS certified or non-FIPS certified CipherSpecs.

If the queue manager runs without using cryptographic hardware, refer to the CipherSpecs listed in Specifying CipherSpecs .

This is the queue manager's initial default value.

YES Specifies that only FIPS-certified algorithms are to be used in the CipherSpecs allowed on all SSL connections from and to this queue manager.

For a listing of appropriate FIPS 140-2 certified CipherSpecs; see Specifying CipherSpecs .

Changes to SSLFIPS become effective either:

- On UNIX, Linux, and Windows systems, when a new channel process is started.
- For channels that run as threads of the channel initiator on UNIX, Linux, and Windows systems, when the channel initiator is restarted.

- For channels that run as threads of the listener on UNIX, Linux, and Windows systems, when the listener is restarted.
- For channels that run as threads of a process pooling process, when the process pooling process is started or restarted and first runs an SSL channel. If the process pooling process has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command `REFRESH SECURITY TYPE(SSL)`. The process pooling process is **amqrmppa** on UNIX, Linux, and Windows systems.
- On z/OS, when the channel initiator is restarted.
- When a`REFRESH SECURITY TYPE(SSL)` command is issued, except on z/OS.

SSLKEYR(*string***)**

The name of the Secure Sockets Layer key repository.

The maximum length of the string is 256 characters.

The format of the name depends on the environment:

- On z/OS, it is the name of a key ring.
- On IBM i, it is of the form *pathname/keyfile*, where *keyfile* is specified without the suffix `.kdb` , and identifies a GSKit key database file.

  If you specify `*SYSTEM`, IBM WebSphere MQ uses the system certificate store as the key repository for the queue manager. The queue manager is registered as a server application in the Digital Certificate Manager (DCM). You can assign any server/client certificate in the system store to the queue manager, because you registered it as a server application.

  If you change the `SSLKEYR` parameter to a value other than `*SYSTEM`, IBM WebSphere MQ unregisters the queue manager as an application with DCM.
- On UNIX and Linux, it is of the form *pathname/keyfile* and on Windows *pathname\keyfile* , where *keyfile* is specified without the suffix `.kdb`, and identifies a GSKit key database file.

On IBM i, UNIX, Linux, and Windows systems, the syntax of this parameter is validated to ensure that it contains a valid, absolute, directory path.

If `SSLKEYR` is blank, channels using SSL fail to start. If `SSLKEYR` is set to a value that does not correspond to a key ring or key database file, channels using SSL also fail to start.

Changes to `SSLKEYR` become effective either:

- On IBM i, UNIX, Linux, and Windows systems, when a new channel process is started.
- For channels that run as threads of the channel initiator on IBM i, UNIX, Linux, and Windows systems, when the channel initiator is restarted.
- For channels that run as threads of the listener on IBM i, UNIX, Linux, and Windows systems, when the listener is restarted.
- For channels that run as threads of a process pooling process, **amqrmppa**, when the process pooling process is started or restarted and first runs an SSL channel. If the process pooling process has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command `REFRESH SECURITY TYPE(SSL)`.
- On z/OS, when the channel initiator is restarted.
- When a `REFRESH SECURITY TYPE(SSL)` command is issued.

SSLRKEYC(*integer***)**

The number of bytes to be sent and received within an SSL conversation before the secret key is renegotiated. The number of bytes includes control information.

`SSLRKEYC` is used only by SSL channels which initiate communication from the queue manager. For example, the sender channel initiates communication in a sender and receiver channel pairing.

If a value greater than zero is specified, the secret key is also renegotiated before message data is sent or received following a channel heartbeat. The count of bytes until the next secret key renegotiation is reset after each successful renegotiation.

Specify a value in the range 0 - 999,999,999. A value of zero means that the secret key is never renegotiated. If you specify an SSL/TLS secret key reset count in the range 1 - 32767 bytes (32 KB), SSL/TLS channels use a secret key reset count of 32 KB. The larger reset count value avoids the cost of excessive key resets which would occur for small SSL/TLS secret key reset values.

Attention: Non-zero values less than 4096 (4 KB) might cause channels to fail to start, or might cause inconsistencies in the values of SSLKEYDA, SSLKEYTI, and SSLRKEYS.

SSLTASKS(*integer***)**
The number of server subtasks to use for processing SSL calls. To use SSL channels, you must have at least two of these tasks running.

This parameter is valid only on z/OS.

This value is in the range 0 - 9999. To avoid problems with storage allocation, do not set the SSLTASKS parameter to a value greater than 50.

Changes to this parameter are effective when the channel initiator is restarted.

STATACLS
Specifies whether statistics data is to be collected for auto-defined cluster-sender channels:

QMGR
Collection of statistics data is inherited from the setting of the STATCHL parameter of the queue manager.

This is the queue manager's initial default value.

OFF Statistics data collection for the channel is switched off.

LOW Unless STATCHL is NONE, statistics data collection is switched on with a low ratio of data collection with a minimal effect on system performance.

MEDIUM
Unless STATCHL is NONE, statistics data collection is switched on with a moderate ratio of data collection.

HIGH Unless STATCHL is NONE, statistics data collection is switched on with a high ratio of data collection.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

STATCHL
Specifies whether statistics data is to be collected for channels:

NONE
Statistics data collection is turned off for channels regardless of the setting of their STATCHL parameter.

OFF Statistics data collection is turned off for channels specifying a value of QMGR in their STATCHL parameter.

This is the queue manager's initial default value.

LOW Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of QMGR in their STATCHL parameter.

MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of QMGR in their STATCHL parameter.

HIGH Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of QMGR in their STATCHL parameter.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

A change to this parameter takes effect only on channels started after the change occurs. Any channel started before the change to the parameter continues with the value in force at the time that the channel started.

STATINT(*integer***)**
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue.

Specify a value in the range 1 through 604800.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

Changes to this parameter take immediate effect on the collection of monitoring and statistics data.

STATMQI
Specifies whether statistics monitoring data is to be collected for the queue manager:

OFF Data collection for MQI statistics is disabled.

This is the queue manager's initial default value.

ON Data collection for MQI statistics is enabled.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

Changes to this parameter take immediate effect on the collection of monitoring and statistics data.

STATQ
Specifies whether statistics data is to be collected for queues:

NONE
Statistics data collection is turned off for queues regardless of the setting of their STATQ parameter.

OFF Statistics data collection is turned off for queues specifying a value of QMGR or OFF in their STATQ parameter. OFF is the default value.

ON Statistics data collection is turned on for queues specifying a value of QMGR or ON in their STATQ parameter.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

Statistics messages are generated only for queues which are opened after statistics collection is enabled. You do not need to restart the queue manager for the new value of STATQ to take effect.

STRSTPEV
Specifies whether start and stop events are generated:

ENABLED
Start and stop events are generated.

This is the queue manager's initial default value.

DISABLED
Start and stop events are not generated.

SUITEB

Specifies whether Suite B-compliant cryptography is used and what strength is required.

NONE

Suite B is not used. NONE is the default

128_BIT

Suite B 128-bit level security is used.

192_BIT

Suite B 192-bit level security is used

128_BIT,192_BIT

Both Suite B 128-bit and 192-bit level security is used

TCPCHL(*integer* **)**

The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol.

The maximum number of sockets used is the sum of the values in TCPCHL and CHIDISPS. The z/OS UNIX System Services MAXFILEPROC parameter (specified in the BPXPRM*xx* member of SYS1.PARMLIB) controls how many sockets each task is allowed, and thus how many channels each dispatcher is allowed. In this case, the number of channels using TCP/IP is limited to the value of MAXFILEPROC multiplied by the value of CHIDISPS.

Specify a value 0-9999. The value must not be greater than the value of MAXCHL . MAXCHL defines the maximum number of channels available. TCP/IP might not support as many as 9999 channels. If so, the value you can specify is limited by the number of channels TCP/IP can support. If you specify zero, the TCP/IP transmission protocol is not used.

If you change this value, also review the MAXCHL, LU62CHL, and ACTCHL values to ensure that there is no conflict of values. If necessary, raise the value of MAXCHL and ACTCHL.

If the value of this parameter is reduced, any current channels that exceed the new limit continue to run until they stop.

Sharing conversations do not contribute to the total for this parameter.

This parameter is valid on z/OS only.

TCPKEEP

Specifies whether the KEEPALIVE facility is to be used to check that the other end of the connection is still available. If it is unavailable, the channel is closed.

NO The TCP KEEPALIVE facility is not to be used.

This is the queue manager's initial default value.

YES The TCP KEEPALIVE facility is to be used as specified in the TCP profile configuration data set. The interval is specified in the KAINT channel attribute.

This parameter is valid on z/OS only.

Changes to this parameter take effect for channels that are later started. Channels that are currently started are unaffected by changes to this parameter.

Using the TCPKEEP parameter is no longer required for 'modern' queue managers. The replacement is a combination of:

- using 'modern' client channels (SHARECNV <> 0); and
- using receive timeout for message channels RCVTIME.

For more information, see the technote "Setting the TCP/IP KeepAlive interval to be used by WebSphere MQ", at the following address: http://www.ibm.com/support/docview.wss?uid=swg21216834.

TCPNAME(*string* **)**
The name of either the only, or default, TCP/IP system to be used, depending on the value of TCPSTACK. This name is the name of the z/OS UNIX System Services stack for TCP/IP, as specified in the SUBFILESYSTYPE NAME parameter in the BPXPRM*xx* member of SYS1.PARMLIB.

The maximum length of this parameter is eight characters.

This parameter is valid on z/OS only.

Changes to this parameter take effect when the channel initiator is restarted.

TCPSTACK
Specifies whether the channel initiator can use only the TCP/IP address space specified in TCPNAME, or optionally bind to any selected TCP/IP address.

SINGLE
The channel initiator can use only the TCP/IP address space specified in TCPNAME.

MULTIPLE
The channel initiator can use any TCP/IP address space available to it.

This parameter is valid on z/OS only.

Changes to this parameter take effect when the channel initiator is restarted.

TRAXSTR
Specifies whether the channel initiator trace starts automatically:

YES Channel initiator trace is to start automatically.

NO Channel initiator trace is not to start automatically.

This parameter is valid on z/OS only.

Changes to this parameter take effect when the channel initiator is restarted. If you want to start or stop channel initiator trace without restarting the channel initiator, use the START TRACE or STOP TRACE commands after starting the channel initiator.

TRAXTBL(*integer* **)**
The size, in megabytes, of the trace data space of the channel initiator.

Specify a value in the range 2 through 2048.

This parameter is valid on z/OS only.

Note:

1. Changes to this parameter take effect immediately; any existing trace table contents are lost.
2. The **CHINIT** trace is stored in a dataspace called qmidCHIN.CSQXTRDS. When you use large z/OS data spaces, ensure that sufficient auxiliary storage is available on your system to support any related z/OS paging activity. You might also need to increase the size of your SYS1.DUMP data sets.

TREELIFE(*integer***)**
The lifetime, in seconds of non-administrative topics.

Non-administrative topics are those topics created when an application publishes to, or subscribes on, a topic string that does not exist as an administrative node. When this non-administrative node no longer has any active subscriptions, this parameter determines how long the queue manager waits before removing that node. Only non-administrative topics that are in use by a durable subscription remain after the queue manager is recycled.

Specify a value in the range 0 through 604000. A value of 0 means that non-administrative topics are not removed by the queue manager.

TRIGINT(*integer* **)**

A time interval expressed in milliseconds.

The TRIGINT parameter is relevant only if the trigger type (TRIGTYPE) is set to FIRST (see "DEFINE QLOCAL" on page 536 for details). In this case trigger messages are normally generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with FIRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every TRIGINT milliseconds; see Special case of trigger type FIRST.

Specify a value in the range 0 - 999,999,999.

ALTER queues:

Use the MQSC **ALTER** command to alter the parameters of a queue. A queue might be a local queue (ALTER QLOCAL), alias queue (ALTER QALIAS), model queue (ALTER QMODEL), a remote queue, a queue-manager alias, or a reply-to queue alias (ALTER QREMOTE).

This section contains the following commands:

- "ALTER QALIAS" on page 409
- "ALTER QLOCAL" on page 410
- "ALTER QMODEL" on page 413
- "ALTER QREMOTE" on page 415

These commands are supported on the following platforms:

UNIX and Linux	Windows
✓	✓

Parameters not specified in the **ALTER** queue commands result in the existing values for those parameters being left unchanged.

Parameter descriptions for ALTER QUEUE

The parameters that are relevant for each type of queue are tabulated in Table 70. Each parameter is described after the table.

Table 70. DEFINE and ALTER QUEUE parameters.

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

Parameter	Local queue	Model queue	Alias queue	Remote queue
ACCTQ	✓	✓		
BOQNAME	✓	✓		
BOTHRESH	✓	✓		
CFSTRUCT	✓	✓		
CLCHNAME	✓	✓		

Table 70. DEFINE and ALTER QUEUE parameters (continued).

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

Parameter	Local queue	Model queue	Alias queue	Remote queue
CLUSNL	✓		✓	✓
CLUSTER	✓		✓	✓
CLWLPRTY	✓		✓	✓
CLWLRANK	✓		✓	✓
CLWLUSEQ	✓			
CMDSCOPE	✓	✓	✓	✓
CUSTOM	✓	✓	✓	✓
DEFBIND	✓		✓	✓
DEFPRESP	✓	✓	✓	✓
DEFPRTY	✓	✓	✓	✓
DEFPSIST	✓	✓	✓	✓
DEFREADA	✓	✓	✓	
DEFSOPT	✓	✓		
DEFTYPE	✓	✓		
DESCR	✓	✓	✓	✓
DISTL	✓	✓		
FORCE	✓		✓	✓
GET	✓	✓	✓	
HARDENBO or **NOHARDENBO**	✓	✓		
INDXTYPE	✓	✓		
INITQ	✓	✓		
LIKE	✓	✓	✓	✓
MAXDEPTH	✓	✓		

Table 70. DEFINE and ALTER QUEUE parameters (continued).

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

Parameter	Local queue	Model queue	Alias queue	Remote queue
MAXMSGL	✓	✓		
MONQ	✓	✓		
MSGDLVSQ	✓	✓		
NOREPLACE	✓	✓	✓	✓
NPMCLASS	✓	✓		
PROCESS	✓	✓		
PROPCTL	✓	✓	✓	
PUT	✓	✓	✓	✓
queue-name	✓	✓	✓	✓
QDEPTHHI	✓	✓		
QDEPTHLO	✓	✓		
QDPHIEV	✓	✓		
QDPLOEV	✓	✓		
QDPMAXEV	✓	✓		
QSGDISP	✓	✓	✓	✓
QSVCIEV	✓	✓		
QSVCINT	✓	✓		
REPLACE	✓	✓	✓	✓
RETINTVL	✓	✓		
RNAME				✓
RQMNAME				✓
SCOPE	✓		✓	✓
SHARE or **NOSHARE**	✓	✓		

Table 70. DEFINE and ALTER QUEUE parameters (continued).

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

Parameter	Local queue	Model queue	Alias queue	Remote queue
STATQ	✓	✓		
STGCLASS	✓	✓		
TARGET			✓	
TARGQ			✓	
TARGTYPE			✓	
TRIGDATA	✓	✓		
TRIGDPTH	✓	✓		
TRIGGER or **NOTRIGGER**	✓	✓		
TRIGMPRI	✓	✓		
TRIGTYPE	✓	✓		
USAGE	✓	✓		
XMITQ				✓

queue-name
Local name of the queue, except the remote queue where it is the local definition of the remote queue.

The name must not be the same as any other queue name of any queue type currently defined on this queue manager, unless you specify **REPLACE** or **ALTER**. See Rules for naming IBM WebSphere MQ objects.

ACCTQ
Specifies whether accounting data collection is to be enabled for the queue. On z/OS, the data collected is class 3 accounting data (thread-level and queue-level accounting). In order for accounting data to be collected for this queue, accounting data for this connection must also be enabled. Turn on accounting data collection by setting either the **ACCTQ** queue manager attribute, or the options field in the MQCNO structure on the MQCONNX call.

QMGR The collection of accounting data is based on the setting of the **ACCTQ** parameter on the queue manager definition.

ON Accounting data collection is enabled for the queue unless the **ACCTQ** queue manager parameter has a value of NONE. On z/OS systems, you must switch on class 3 accounting using the **START TRACE** command.

OFF Accounting data collection is disabled for the queue.

BOQNAME(*queue-name***)**
The excessive backout requeue name.

This parameter is supported only on local and model queues.

Use this parameter to set or change the back out queue name attribute of a local or model queue. Apart from allowing its value to be queried, the queue manager does nothing based on the value of this attribute. IBM WebSphere MQ classes for JMS transfers a message that is backed out the maximum number of times to this queue. The maximum is specified by the **BOTHRESH** attribute.

BOTHRESH(*integer***)**
The backout threshold.

This parameter is supported only on local and model queues.

Use this parameter to set or change the value of the back out threshold attribute of a local or model queue. Apart from allowing its value to be queried, the queue manager does nothing based on the value of this attribute. IBM WebSphere MQ classes for JMS use the attribute to determine how many times back a message out. When the value is exceeded the message is transferred to the queue named by the **BOQNAME** attribute.

Specify a value in the range 0 - 999,999,999.

CFSTRUCT(*structure-name***)**
Specifies the name of the coupling facility structure where you want messages stored when you use shared queues.

This parameter is supported only on z/OS for local and model queues.

The name:

- Cannot have more than 12 characters
- Must start with an uppercase letter (A - Z)
- Can include only the characters A - Z and 0 - 9

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always four characters, padded with @ symbols if necessary. For example, if you use a queue-sharing group named NY03 and you supply the name PRODUCT7, the resultant coupling facility structure name is NY03PRODUCT7. The administrative structure for the queue-sharing group (in this case NY03CSQ_ADMIN) cannot be used for storing messages.

For ALTER QLOCAL, ALTER QMODEL, DEFINE QLOCAL with **REPLACE**, and DEFINE QMODEL with **REPLACE** the following rules apply:

- On a local queue with **QSGDISP**(SHARED), **CFSTRUCT** cannot change.

  If you change either the **CFSTRUCT** or **QSGDISP** value you must delete and redefine the queue. To preserve any of the messages on the queue you must offload the messages before you delete the queue. Reload the messages after you redefine the queue, or move the messages to another queue.
- On a model queue with **DEFTYPE**(SHAREDYN), **CFSTRUCT** cannot be blank.
- On a local queue with a **QSGDISP** other than SHARED, or a model queue with a **DEFTYPE** other than SHAREDYN, the value of **CFSTRUCT** does not matter.

For DEFINE QLOCAL with **NOREPLACE** and DEFINE QMODEL with **NOREPLACE**, the coupling facility structure:

- On a local queue with **QSGDISP**(SHARED) or a model queue with a **DEFTYPE**(SHAREDYN), **CFSTRUCT** cannot be blank.
- On a local queue with a **QSGDISP** other than SHARED, or a model queue with a **DEFTYPE** other than SHAREDYN, the value of **CFSTRUCT** does not matter.

Note: Before you can use the queue, the structure must be defined in the coupling facility Resource Management (CFRM) policy data set.

CLCHNAME(*channel name***)**

This parameter is supported only on transmission queues.

CLCHNAME is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue. CLCHNAME is not supported on z/OS.

You can also set the transmission queue attribute CLCHNAME attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the CLCHNAME attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either SYSTEM.CLUSTER.TRANSMIT.*ChannelName* or SYSTEM.CLUSTER.TRANSMIT.QUEUE, depending on the value of the queue manager DEFCLXQ attribute.

By specifying asterisks, “"*"”, in CLCHNAME, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. CLCHNAME is limited to a length of 48 characters, MQ_OBJECT_NAME_LENGTH. A channel name is limited to 20 characters: MQ_CHANNEL_NAME_LENGTH.

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. The default configuration can be changed by modified by changing the queue manager attribute, DEFCLXQ. The default value of the attribute is SCTQ. You can change the value to CHANNEL. If you set the DEFCLXQ attribute to CHANNEL, each cluster-sender channel defaults to using a specific cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

CLUSNL(*namelist name***)**
The name of the namelist that specifies a list of clusters to which the queue belongs.

This parameter is supported only on alias, local, and remote queues.

Changes to this parameter do not affect instances of the queue that are already open.

Only one of the resultant values of **CLUSNL** or **CLUSTER** can be nonblank; you cannot specify a value for both.

On local queues, this parameter cannot be set for transmission, SYSTEM.CHANNEL.*xx*, SYSTEM.CLUSTER.*xx*, or SYSTEM.COMMAND.*xx* queues, and on z/OS only, for SYSTEM.QSG.*xx* queues.

This parameter is valid only on AIX, HP-UX, Linux, Solaris, Windows, and z/OS.

CLUSTER(*cluster name***)**
The name of the cluster to which the queue belongs.

This parameter is supported only on alias, local, and remote queues.

The maximum length is 48 characters conforming to the rules for naming IBM WebSphere MQ objects. Changes to this parameter do not affect instances of the queue that are already open.

Only one of the resultant values of **CLUSNL** or **CLUSTER** can be nonblank; you cannot specify a value for both.

On local queues, this parameter cannot be set for transmission, SYSTEM.CHANNEL.*xx*, SYSTEM.CLUSTER.*xx*, or SYSTEM.COMMAND.*xx* queues, and on z/OS only, for SYSTEM.QSG.*xx* queues.

This parameter is valid only on AIX, HP-UX, Linux, Solaris, Windows, and z/OS.

CLWLPRTY(*integer***)**
Specifies the priority of the queue for the purposes of cluster workload distribution. This parameter is valid only for local, remote, and alias queues. The value must be in the range zero through 9 where zero is the lowest priority and 9 is the highest. For more information about this attribute, see CLWLPRTY queue attribute.

CLWLRANK(*integer***)**
Specifies the rank of the queue for the purposes of cluster workload distribution. This parameter

is valid only for local, remote, and alias queues. The value must be in the range zero through 9 where zero is the lowest rank and 9 is the highest. For more information about this attribute, see CLWLRANK queue attribute.

CLWLUSEQ
Specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance. The parameter has no effect when the MQPUT originates from a cluster channel. This parameter is valid only for local queues.

QMGR The behavior is as specified by the **CLWLUSEQ** parameter of the queue manager definition.

ANY The queue manager is to treat the local queue as another instance of the cluster queue for the purposes of workload distribution.

LOCAL The local queue is the only target of the MQPUT operation.

CMDSCOPE
This parameter applies to z/OS only. It specifies where the command is run when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP or SHARED.

'' The command is run on the queue manager on which it was entered.

QmgrName
The command is run on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered. You can specify another name, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

CUSTOM(*string***)**
The custom attribute for new features.

This attribute is reserved for the configuration of new features before separate attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

This description is updated when features using this attribute are introduced. At the moment, there are no values for **CUSTOM**.

DEFBIND
Specifies the binding to be used when the application specifies MQOO_BIND_AS_Q_DEF on the MQOPEN call, and the queue is a cluster queue.

OPEN The queue handle is bound to a specific instance of the cluster queue when the queue is opened.

NOTFIXED
The queue handle is not bound to any instance of the cluster queue. The queue manager selects a specific queue instance when the message is put using MQPUT. It changes that selection later, if the need arises.

GROUP Allows an application to request that a group of messages is allocated to the same destination instance.

Multiple queues with the same name can be advertised in a queue manager cluster. An application can send all messages to a single instance, MQOO_BIND_ON_OPEN. It can allow a workload management algorithm to select the most suitable destination on a per message basis,

MQOO_BIND_NOT_FIXED. It can allow an application to request that a "group" of messages be all allocated to the same destination instance. The workload balancing reselects a destination between groups of messages, without requiring an MQCLOSE and MQOPEN of the queue.

The MQPUT1 call always behaves as if NOTFIXED is specified.

This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

DEFPRESP
Specifies the behavior to be used by applications when the put response type, within the MQPMO options, is set to MQPMO_RESPONSE_AS_Q_DEF.

SYNC Put operations to the queue specifying MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_SYNC_RESPONSE is specified instead.

ASYNC Put operations to the queue specifying MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_ASYNC_RESPONSE is specified instead; see MQPMO options (MQLONG).

DEFPRTY(*integer*)
The default priority of messages put on the queue. The value must be in the range 0 - 9. Zero is the lowest priority, through to the **MAXPRTY** queue manager parameter. The default value of **MAXPRTY** is 9.

DEFPSIST
Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_Q_DEF option.

NO Messages on this queue are lost across a restart of the queue manager.

YES Messages on this queue survive a restart of the queue manager.

On z/OS, N and Y are accepted as synonyms of NO and YES.

DEFREADA
Specifies the default read ahead behavior for non-persistent messages delivered to the client. Enabling read ahead can improve the performance of client applications consuming non-persistent messages.

NO Non-persistent messages are not read ahead unless the client application is configured to request read ahead.

YES Non-persistent messages are sent to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not delete all the messages it is sent.

DISABLED
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

DEFSOPT
The default share option for applications opening this queue for input:

EXCL The open request is for exclusive input from the queue

SHARED The open request is for shared input from the queue

DEFTYPE
Queue definition type.

This parameter is supported only on model queues.

PERMDYN
A permanent dynamic queue is created when an application issues an MQOPEN MQI call with the name of this model queue specified in the object descriptor (MQOD).

On z/OS, the dynamic queue has a disposition of QMGR.

SHAREDYN
This option is available on z/OS only.

A permanent dynamic queue is created when an application issues an MQOPEN API call with the name of this model queue specified in the object descriptor (MQOD).

The dynamic queue has a disposition of SHARED.

TEMPDYN
A temporary dynamic queue is created when an application issues an MQOPEN API call with the name of this model queue specified in the object descriptor (MQOD).

On z/OS, the dynamic queue has a disposition of QMGR.

Do not specify this value for a model queue definition with a **DEFPSIST** parameter of YES.

If you specify this option, do not specify **INDXTYPE**(MSGTOKEN).

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY QUEUE command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters that are in the coded character set identifier (CCSID) of this queue manager. If you do not do so and if the information is sent to another queue manager, they might be translated incorrectly.

DISTL
DISTL sets whether distribution lists are supported by the partner queue manager.

YES Distribution lists are supported by the partner queue manager.

NO Distribution lists are not supported by the partner queue manager.

Note: You do not normally change this parameter, because it is set by the MCA. However you can set this parameter when defining a transmission queue if the distribution list capability of the destination queue manager is known.

This parameter is valid only on AIX, HP-UX, Linux, Solaris, and Windows.

FORCE
This parameter applies only to the ALTER command on alias, local and remote queues.

Specify this parameter to force completion of the command in the following circumstances.

For an alias queue, if both of the following are true:

- The **TARGQ** parameter is specified
- An application has this alias queue open

For a local queue, if both of the following are true:

- The **NOSHARE** parameter is specified
- More than one application has the queue open for input

FORCE is also needed if both of the following are true:

- The **USAGE** parameter is changed
- Either one or more messages are on the queue, or one or more applications have the queue open

Do not change the **USAGE** parameter while there are messages on the queue; the format of messages changes when they are put on a transmission queue.

For a remote queue, if both of the following are true:

- The **XMITQ** parameter is changed
- One or more applications has this queue open as a remote queue

FORCE is also needed if both of the following are true:

- Any of the **RNAME**, **RQMNAME**, or **XMITQ** parameters are changed
- One or more applications has a queue open that resolved through this definition as a queue manager alias

Note: **FORCE** is not required if this definition is in use as a reply-to queue alias only.

If **FORCE** is not specified in the circumstances described, the command is unsuccessful.

GET Specifies whether applications are to be permitted to get messages from this queue:

ENABLED
Messages can be retrieved from the queue, by suitably authorized applications.

DISABLED
Applications cannot retrieve messages from the queue.

This parameter can also be changed using the MQSET API call.

HARDENBO&NOHARDENBO
Specifies whether hardening is used to ensure that the count of the number of times that a message is backed out is accurate.

This parameter is supported only on local and model queues.

HARDENBO
The count is hardened.

NOHARDENBO
The count is not hardened.

Note: This parameter affects only IBM WebSphere MQ for z/OS. It can be set on other platforms but is ineffective.

INDXTYPE
The type of index maintained by the queue manager to expedite MQGET operations on the queue. For shared queues, the type of index determines the type of MQGET operations that can be used.

This parameter is supported only on local and model queues.

Messages can be retrieved using a selection criterion only if an appropriate index type is maintained, as the following table shows:

Index type required for different retrieval selection criteria

Retrieval selection criterion	**Index type required**	
	Shared queue	Other queue
None (sequential retrieval)	Any	Any
Message identifier	MSGID or NONE	Any
Correlation identifier	CORRELID	Any
Message and correlation identifiers	MSGID or CORRELID	Any
Group identifier	GROUPID	Any
Grouping	GROUPID	GROUPID

Index type required for different retrieval selection criteria

Retrieval selection criterion	Index type required	
Message token	Not allowed	MSGTOKEN

where the value of **INDXTYPE** parameter has the following values:

NONE No index is maintained. Use NONE when messages are typically retrieved sequentially or use both the message identifier and the correlation identifier as a selection criterion on the MQGET call.

MSGID An index of message identifiers is maintained. Use MSGID when messages are typically retrieved using the message identifier as a selection criterion on the MQGET call with the correlation identifier set to NULL.

CORRELID
An index of correlation identifiers is maintained. Use CORRELID when messages are typically retrieved using the correlation identifier as a selection criterion on the MQGET call with the message identifier set to NULL.

GROUPID
An index of group identifiers is maintained. Use GROUPID when messages are retrieved using message grouping selection criteria.

Note:

1. You cannot set **INDXTYPE** to GROUPID if the queue is a transmission queue.
2. The queue must use a CF structure at CFLEVEL(3), to specify a shared queue with **INDXTYPE**(GROUPID).

MSGTOKEN
An index of message tokens is maintained. Use MSGTOKEN when the queue is a WLM-managed queue that you are using with the Workload Manager functions of z/OS.

Note: You cannot set **INDXTYPE** to MSGTOKEN if:

- The queue is a model queue with a definition type of SHAREDYN
- The queue is a temporary dynamic queue
- The queue is a transmission queue
- You specify **QSGDISP**(SHARED)

For queues that are not shared and do not use grouping or message tokens, the index type does not restrict the type of retrieval selection. However, the index is used to expedite **GET** operations on the queue, so choose the type that corresponds to the most common retrieval selection.

If you are altering or replacing an existing local queue, you can change the **INDXTYPE** parameter only in the cases indicated in the following table:

Index type change permitted depending upon queue sharing and presence of messages in the queue.

Queue type		NON-SHARED			SHARED	
Queue state		Uncommitted activity	No uncommitted activity, messages present	No uncommitted activity, and empty	Open or messages present	Not open, and empty
Change INDXTYPE from:	To:	Change allowed?				
NONE	MSGID	No	Yes	Yes	No	Yes

Index type change permitted depending upon queue sharing and presence of messages in the queue.

Queue type		NON-SHARED			SHARED	
NONE	CORRELID	No	Yes	Yes	No	Yes
NONE	MSGTOKEN	No	No	Yes	-	-
NONE	GROUPID	No	No	Yes	No	Yes
MSGID	NONE	No	Yes	Yes	No	Yes
MSGID	CORRELID	No	Yes	Yes	No	Yes
MSGID	MSGTOKEN	No	No	Yes	-	-
MSGID	GROUPID	No	No	Yes	No	Yes
CORRELID	NONE	No	Yes	Yes	No	Yes
CORRELID	MSGID	No	Yes	Yes	No	Yes
CORRELID	MSGTOKEN	No	No	Yes	-	-
CORRELID	GROUPID	No	No	Yes	No	Yes
MSGTOKEN	NONE	No	Yes	Yes	-	-
MSGTOKEN	MSGID	No	Yes	Yes	-	-
MSGTOKEN	CORRELID	No	Yes	Yes	-	-
MSGTOKEN	GROUPID	No	No	Yes	-	-
GROUPID	NONE	No	No	Yes	No	Yes
GROUPID	MSGID	No	No	Yes	No	Yes
GROUPID	CORRELID	No	No	Yes	No	Yes
GROUPID	MSGTOKEN	No	No	Yes	-	-

This parameter is supported only on z/OS. On other platforms, all queues are automatically indexed.

INITQ(*string***)**
The local name of the initiation queue on this queue manager, to which trigger messages relating to this queue are written; see Rules for naming IBM WebSphere MQ objects .

This parameter is supported only on local and model queues.

LIKE(*qtype-name***)**
The name of a queue, with parameters that are used to model this definition.

If this field is not completed, the values of undefined parameter fields are taken from one of the following definitions. The choice depends on the queue type:

Queue type	Definition
Alias queue	SYSTEM.DEFAULT.ALIAS.QUEUE
Local queue	SYSTEM.DEFAULT.LOCAL.QUEUE
Model queue	SYSTEM.DEFAULT.MODEL.QUEUE
Remote queue	SYSTEM.DEFAULT.REMOTE.QUEUE

For example, not completing this parameter is equivalent to defining the following value of LIKE for an alias queue:

```
LIKE(SYSTEM.DEFAULT.ALIAS.QUEUE)
```

If you require different default definitions for all queues, alter the default queue definitions instead of using the **LIKE** parameter.

On z/OS, the queue manager searches for an object with the name and queue type you specify with a disposition of QMGR, COPY, or SHARED. The disposition of the **LIKE** object is not copied to the object you are defining.

Note:

1. **QSGDISP** (GROUP) objects are not searched.
2. **LIKE** is ignored if **QSGDISP**(COPY) is specified.

MAXDEPTH(*integer***)**
The maximum number of messages allowed on the queue.

This parameter is supported only on local and model queues.

On AIX, HP-UX, Linux, Solaris, Windows, and z/OS, specify a value in the range zero through 999999999.

This parameter is valid only on AIX, HP-UX, Linux, Solaris, Windows, and z/OS.

On any other IBM WebSphere MQ platform, specify a value in the range zero through 640000.

Other factors can still cause the queue to be treated as full, for example, if there is no further hard disk space available.

If this value is reduced, any messages that are already on the queue that exceed the new maximum remain intact.

MAXMSGL(*integer***)**
The maximum length (in bytes) of messages on this queue.

This parameter is supported only on local and model queues.

On AIX, HP-UX, Linux, Solaris, and Windows, specify a value in the range zero to the maximum message length for the queue manager. See the **MAXMSGL** parameter of the ALTER QMGR command, ALTER QMGR MAXMSGL.

On z/OS, specify a value in the range zero through 100 MB (104 857 600 bytes).

Message length includes the length of user data and the length of headers. For messages put on the transmission queue, there are additional transmission headers. Allow an additional 4000 bytes for all the message headers.

If this value is reduced, any messages that are already on the queue with length that exceeds the new maximum are not affected.

Applications can use this parameter to determine the size of buffer for retrieving messages from the queue. Therefore, the value can be reduced only if it is known that this reduction does not cause an application to operate incorrectly.

Note that by adding the digital signature and key to the message, WebSphere MQ Advanced Message Security increases the length of the message.

MONQ
Controls the collection of online monitoring data for queues.

This parameter is supported only on local and model queues.

QMGR Collect monitoring data according to the setting of the queue manager parameter **MONQ**.

OFF Online monitoring data collection is turned off for this queue.

LOW If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

MEDIUM If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

HIGH If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

There is no distinction between the values LOW, MEDIUM, and HIGH. These values all turn data collection on, but do not affect the rate of collection.

When this parameter is used in an ALTER queue command, the change is effective only when the queue is next opened.

MSGDLVSQ
Message delivery sequence.

This parameter is supported only on local and model queues.

PRIORITY
Messages are delivered (in response to MQGET API calls) in first-in-first-out (FIFO) order within priority.

FIFO Messages are delivered (in response to MQGET API calls) in FIFO order. Priority is ignored for messages on this queue.

The message delivery sequence parameter can be changed from PRIORITY to FIFO while there are messages on the queue. The order of the messages already on the queue is not changed. Messages added to the queue later take the default priority of the queue, and so might be processed before some of the existing messages.

If the message delivery sequence is changed from FIFO to PRIORITY, the messages put on the queue while the queue was set to FIFO take the default priority.

Note: If **INDXTYPE**(GROUPID) is specified with **MSGDLVSQ**(PRIORITY), the priority in which groups are retrieved is based on the priority of the first message within each group. The priorities 0 and 1 are used by the queue manager to optimize the retrieval of messages in logical order. The first message in each group must not use these priorities. If it does, the message is stored as if it was priority two.

NPMCLASS
The level of reliability to be assigned to non-persistent messages that are put to the queue:

NORMAL Non-persistent messages are lost after a failure, or queue manager shutdown. These messages are discarded on a queue manager restart.

HIGH The queue manager attempts to retain non-persistent messages on this queue over a queue manager restart or switch over.

You cannot set this parameter on z/OS.

PROCESS(*string***)**
The local name of the IBM WebSphere MQ process.

This parameter is supported only on local and model queues.

This parameter is the name of a process instance that identifies the application started by the queue manager when a trigger event occurs; see Rules for naming IBM WebSphere MQ objects .

The process definition is not checked when the local queue is defined, but it must be available for a trigger event to occur.

If the queue is a transmission queue, the process definition contains the name of the channel to be started. This parameter is optional for transmission queues on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS. If you do not specify it, the channel name is taken from the value specified for the **TRIGDATA** parameter.

PROPCTL
Property control attribute. The attribute is optional. It is applicable to local, alias, and model

queues. For information about how an administrator can use **PROPCTL** to control applications that access message properties, see Using the **PROPCTL** queue property to control message properties. Using the **PROPCTL** channel property to control message properties explains how to use **PROPCTL** to control the transfer of properties to queue managers running at Version 6.0 or earlier.

PROPCTL options are as follows. The options do not affect message properties in the MQMD or MQMD extension.

ALL

Set ALL so that an application can read all the properties of the message either in MQRFH2 headers, or as properties of the message handle.

The ALL option enables applications that cannot be changed to access all the message properties from MQRFH2 headers. Applications that can be changed, can access all the properties of the message as properties of the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

COMPAT

Set COMPAT so that unmodified applications that expect JMS-related properties to be in an MQRFH2 header in the message data continue to work as before. Applications that can be changed, can access all the properties of the message as properties of the message handle.

If the message contains a property with a prefix of mcd., jms., usr., or mqext., all message properties are delivered to the application. If no message handle is supplied, properties are returned in an MQRFH2 header. If a message handle is supplied, all properties are returned in the message handle.

If the message does not contain a property with one of those prefixes, and the application does not provide a message handle, no message properties are returned to the application. If a message handle is supplied, all properties are returned in the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

FORCE

Force all applications to read message properties from MQRFH2 headers.

Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible using the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

NONE

If a message handle is supplied, all the properties are returned in the message handle.

All message properties are removed from the message body before it is delivered to the application.

V6COMPAT

Set V6COMPAT so that applications that expect to receive the same MQRFH2 created by a sending application, can receive it as it was sent. The data in the MQRFH2 header is subject to character set conversion and numeric encoding changes. If the application sets properties using MQSETMP, the properties are not added to the MQRFH2 header created by

the application. The properties are accessible only by using the MQINQMP call. The properties are transmitted in an extra MQRFH2 that is visible to channel exits, but not to MQI programs. If properties are inserted in the MQRFH2 header by the sending application, they are only accessible to the receiving application in the MQRFH2 header. You cannot query properties set this way by calling MQINQMP. This behavior of properties and application created MQRFH2 headers occurs only when V6COMPAT is set.

The receiving application can override the setting of V6COMPAT, by setting an MQGMO_PROPERTIES option, such as MQGMO_PROPERTIES_IN_HANDLE. The default setting of MQGMO_PROPERTIES is MQGMO_PROPERTIES_AS_Q_DEF, which leaves the property setting as defined by the **PROPCTL** setting on the resolved receiving queue.

Note: If the **PSPROP** subscription attribute is set to RFH2, the queue manager might add publish/subscribe properties to the psc folder in the application-created MQRFH2 header. Otherwise, the queue manager does not modify the application-created MQRFH2 header.

Special rules apply to setting V6COMPAT:

1. You must set V6COMPAT on both of the queues accessed by MQPUT and MQGET.
   - You might find the effect of V6COMPAT does not require setting V6COMPAT on the queue that MQPUT writes to. The reason is that in many cases MQPUT does not reorganize the contents of an MQRFH2. Setting V6COMPAT has no apparent effect.
   - V6COMPAT appears to take effect only when it is set on the queue that is accessed by the application receiving the message.

   Despite these appearances, it is important you set V6COMPAT for both the sender and receiver of a message. In some circumstances, V6COMPAT works only if it is set at both ends of the transfer.
2. If you set V6COMPAT on either an alias queue or a local queue, the result is the same.

   For example, an alias queue, QA1, has a target queue Q1. An application opens QA1. Whichever of the pairs of definitions in Figure 5 are set, the result is the same. A message is placed on Q1, with the MQRFH2 created by the application preserved exactly as it was when it was passed to the queue manager.

```
DEFINE QLOCAL(Q1)  PROPCTL(V6COMPAT)
DEFINE QALIAS(QA1) TARGET(Q1)

DEFINE QLOCAL(Q1)
DEFINE QALIAS(QA1) TARGET(Q1) PROPCTL(V6COMPAT)
```

Figure 5. Equivalent definitions of V6COMPAT

3. You can set V6COMPAT on the transmission queue, or a queue that resolves to a transmission queue. The result is to transmit any MQRFH2 in a message exactly as it was created by an application. You cannot set V6COMPAT on a QREMOTE definition.

   No other **PROPCTL** queue options behave this way. To control the way message properties are transmitted to a queue manager running IBM WebSphere MQ Version 6.0 or earlier, set the **PROPCTL** channel attribute; see Using the **PROPCTL** channel property to control message properties.
4. For publish/subscribe, V6COMPAT must be set on a queue that resolves to the destination for a publication.
   - For unmanaged publish/subscribe, set V6COMPAT on a queue that is in the name resolution path for the queue passed to MQSUB. If a subscription is created administratively, set V6COMPAT on a queue that is in the name resolution path for the destination set for the subscription.

- For managed publish/subscribe, set V6COMPAT on the model managed durable and managed non-durable queues for subscription topics. The default model managed queues are SYSTEM.MANAGED.DURABLE and SYSTEM.MANAGED.NDURABLE. By using different model queues for different topics, some publications are received with their original MQRFH2, and others with message property control set by other values of **PROPCTL**.
- For queued publish/subscribe, you must identify the queues used by the publishing and subscribing applications. Set V6COMPAT on those queues, as if the publisher and subscriber are using point to point messaging.

The effect of setting V6COMPAT on a message sent to another queue manager is as follows:

To a Version 7.1 queue manager

If a message contains internally set message properties, or message properties set by MQSETMP, the local queue manager adds an MQRFH2. The additional MQRFH2 is placed before any application created MQRFH2 headers. The local queue manager passes the modified message to the channel.

The new MQRFH2 header is flagged MQRFH_INTERNAL (X'8000000') in the MQRFH2 Flags field; see Flags (MQLONG) .

The channel message, and send and receive exits, are passed the entire message including the additional MQRFH2.

The action of the remote channel depends on whether V6COMPAT is set for the target queue. If it is set, then the internally set properties in the initial MQRFH2 are available to an application in the message handle. The application created MQRFH2 is received unchanged, except for character conversion and numeric encoding transformations.

To a Version 7.0.1 queue manager
Internally set properties are discarded. The MQRFH2 header is transferred unmodified.

To a Version 6.0 or earlier queue manager
Internally set properties are discarded. The MQRFH2 header is transferred unmodified. **PROPCTL** channel options are applied after internally set properties are discarded.

PUT Specifies whether messages can be put on the queue.

ENABLED
Messages can be added to the queue (by suitably authorized applications).

DISABLED
Messages cannot be added to the queue.

This parameter can also be changed using the MQSET API call.

QDEPTHHI(*integer***)**
The threshold against which the queue depth is compared to generate a Queue Depth High event.

This parameter is supported only on local and model queues.

This event indicates that an application put a message on a queue resulting in the number of messages on the queue becoming greater than or equal to the queue depth high threshold. See the **QDPHIEV** parameter.

The value is expressed as a percentage of the maximum queue depth (**MAXDEPTH** parameter), and must be in the range zero through 100 and no less than **QDEPTHLO**.

QDEPTHLO(*integer*)
The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This parameter is supported only on local and model queues.

This event indicates that an application retrieved a message from a queue resulting in the number of messages on the queue becoming less than or equal to the queue depth low threshold. See the **QDPLOEV** parameter.

The value is expressed as a percentage of the maximum queue depth (**MAXDEPTH** parameter), and must be in the range zero through 100 and no greater than **QDEPTHHI**.

QDPHIEV
Controls whether Queue Depth High events are generated.

This parameter is supported only on local and model queues.

A Queue Depth High event indicates that an application put a message on a queue resulting in the number of messages on the queue becoming greater than or equal to the queue depth high threshold. See the **QDEPTHHI** parameter.

Note: The value of this parameter can change implicitly, and shared queues on z/OS affect the event. See the description of the Queue Depth High event in Queue Depth High.

ENABLED
Queue Depth High events are generated

DISABLED
Queue Depth High events are not generated

QDPLOEV
Controls whether Queue Depth Low events are generated.

This parameter is supported only on local and model queues.

A Queue Depth Low event indicates that an application retrieved a message from a queue resulting in the number of messages on the queue becoming less than or equal to the queue depth low threshold. See the **QDEPTHLO** parameter.

Note: The value of this parameter can change implicitly. For more information about this event, and the effect that shared queues on z/OS have on this event, see Queue Depth Low .

ENABLED
Queue Depth Low events are generated

DISABLED
Queue Depth Low events are not generated

QDPMAXEV
Controls whether Queue Full events are generated.

This parameter is supported only on local and model queues.

A Queue Full event indicates that a put to a queue was rejected because the queue is full. The queue depth reached its maximum value.

Note: The value of this parameter can change implicitly. For more information about this event, and the effect that shared queues on z/OS have on this event, see Queue Full.

ENABLED
Queue Full events are generated

DISABLED
Queue Full events are not generated

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object within the group.

Action of ALTER depending on different values of **QSGDISP**.

QSGDISP	**ALTER**
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP**(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters **QSGDISP**(QMGR), is not affected by this command.

GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP**(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object), or any object defined using a command that had the parameters **QSGDISP**(SHARED), is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero:
	`DEFINE QUEUE(`*QNAME*`)`
	`REPLACE QSGDISP(COPY)`
	
	The ALTER for the group object takes effect regardless of whether the generated command with **QSGDISP**(COPY) fails.

PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with **QSGDISP**(QMGR) or **QSGDISP**(COPY). Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters **QSGDISP**(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.
SHARED	This value applies only to local queues. The object definition resides in the shared repository. The object was defined using a command that had the parameters **QSGDISP**(SHARED). Any object residing on the page set of the queue manager that executes the command, or any object defined using a command that had the parameters **QSGDISP**(GROUP), is not affected by this command. If the queue is clustered, a command is generated and sent to all active queue managers in the queue-sharing group to notify them of this clustered, shared queue.

QSVCIEV

Controls whether Service Interval High or Service Interval OK events are generated.

This parameter is supported only on local and model queues and is ineffective if it is specified on a shared queue.

A Service Interval High event is generated when a check indicates that no messages were retrieved from the queue for at least the time indicated by the **QSVCINT** parameter.

A Service Interval OK event is generated when a check indicates that messages were retrieved from the queue within the time indicated by the **QSVCINT** parameter.

Note: The value of this parameter can change implicitly. For more information, see the description of the Service Interval High and Service Interval OK events in Queue Service Interval High and Queue Service Interval OK.

HIGH Service Interval High events are generated

OK Service Interval OK events are generated

NONE No service interval events are generated

QSVCINT(*integer***)**
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

This parameter is supported only on local and model queues and is ineffective if it is specified on a shared queue.

See the **QSVCIEV** parameter.

The value is in units of milliseconds, and must be in the range zero through 999999999.

REPLACE & **NOREPLACE**
This option controls whether any existing definition, and on IBM WebSphere MQ for z/OS of the same disposition, is to be replaced with this one. Any object with a different disposition is not changed.

REPLACE
If the object does exist, the effect is like issuing the **ALTER** command without the **FORCE** parameter and with all the other parameters specified. In particular, note that any messages that are on the existing queue are retained.

There is a difference between the **ALTER** command without the **FORCE** parameter, and the **DEFINE** command with the **REPLACE** parameter. The difference is that **ALTER** does not change unspecified parameters, but **DEFINE** with **REPLACE** sets all the parameters. If you use **REPLACE**, unspecified parameters are taken either from the object named on the **LIKE** parameter, or from the default definition, and the parameters of the object being replaced, if one exists, are ignored.

The command fails if both of the following are true:

- The command sets parameters that would require the use of the **FORCE** parameter if you were using the ALTER command
- The object is open

The **ALTER** command with the **FORCE** parameter succeeds in this situation.

If **SCOPE(CELL)** is specified on UNIX and Linux systems, or Windows, and there is already a queue with the same name in the cell directory, the command fails, even if **REPLACE** is specified.

NOREPLACE
The definition must not replace any existing definition of the object.

RETINTVL(*integer***)**
The number of hours from when the queue was defined, after which the queue is no longer needed. The value must be in the range 0 - 999,999,999.

This parameter is supported only on local and model queues.

The CRDATE and CRTIME can be displayed using the **DISPLAY QUEUE** command.

This information is available for use by an operator or a housekeeping application to delete queues that are no longer required.

Note: The queue manager does not delete queues based on this value, nor does it prevent queues from being deleted if their retention interval is not expired. It is the responsibility of the user to take any required action.

RNAME(*string***)**
Name of remote queue. This parameter is the local name of the queue as defined on the queue manager specified by **RQMNAME**.

This parameter is supported only on remote queues.

- If this definition is used for a local definition of a remote queue, **RNAME** must not be blank when the open occurs.
- If this definition is used for a queue manager alias definition, **RNAME** must be blank when the open occurs.

  In a queue manager cluster, this definition applies only to the queue manager that made it. To advertise the alias to the whole cluster, add the **CLUSTER** attribute to the remote queue definition.
- If this definition is used for a reply-to queue alias, this name is the name of the queue that is to be the reply-to queue.

The name is not checked to ensure that it contains only those characters normally allowed for queue names; see Rules for naming IBM WebSphere MQ objects .

RQMNAME(*string***)**

The name of the remote queue manager on which the queue **RNAME** is defined.

This parameter is supported only on remote queues.

- If an application opens the local definition of a remote queue, **RQMNAME** must not be blank or the name of the local queue manager. When the open occurs, if **XMITQ** is blank there must be a local queue of this name, which is to be used as the transmission queue.
- If this definition is used for a queue manager alias, **RQMNAME** is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if **XMITQ** is blank, when the open occurs there must be a local queue of this name, which is to be used as the transmission queue.
- If **RQMNAME** is used for a reply-to queue alias, **RQMNAME** is the name of the queue manager that is to be the reply-to queue manager.

The name is not checked to ensure that it contains only those characters normally allowed for IBM WebSphere MQ object names; see Rules for naming IBM WebSphere MQ objects.

SCOPE

Specifies the scope of the queue definition.

This parameter is supported only on alias, local, and remote queues.

QMGR The queue definition has queue manager scope. This means that the definition of the queue does not extend beyond the queue manager that owns it. You can open a queue for output that is owned by another queue manager in either of two ways:

1. Specify the name of the owning queue manager.
2. Open a local definition of the queue on the other queue manager.

CELL The queue definition has cell scope. Cell scope means that the queue is known to all the queue managers in the cell. A queue with cell scope can be opened for output merely by specifying the name of the queue. The name of the queue manager that owns the queue need not be specified.

If there is already a queue with the same name in the cell directory, the command fails. The **REPLACE** option does not affect this situation.

This value is valid only if a name service supporting a cell directory is configured.

Restriction: The DCE name service is no longer supported.

This parameter is valid only on UNIX and Linux systems, and Windows.

SHARE and NOSHARE

Specifies whether multiple applications can get messages from this queue.

This parameter is supported only on local and model queues.

SHARE More than one application instance can get messages from the queue.

NOSHARE
Only a single application instance can get messages from the queue.

STATQ
Specifies whether statistics data collection is enabled:

QMGR Statistics data collection is based on the setting of the **STATQ** parameter of the queue manager.

ON If the value of the **STATQ** parameter of the queue manager is not NONE, statistics data collection for the queue is enabled.

OFF Statistics data collection for the queue is disabled.

If this parameter is used in an **ALTER** queue command, the change is effective only for connections to the queue manager made after the change to the parameter.

This parameter is valid only on IBM i, UNIX and Linux systems, and Windows.

STGCLASS(*string***)**
The name of the storage class.

This parameter is supported only on local and model queues.

This parameter is an installation-defined name.

This parameter is valid on z/OS only.

The first character of the name must be uppercase A through Z, and subsequent characters either uppercase A through Z or numeric 0 through 9.

Note: You can change this parameter only if the queue is empty and closed.

If you specify **QSGDISP**(SHARED) or **DEFTYPE**(SHAREDYN), this parameter is ignored.

TARGET(*string***)**
The name of the queue or topic object being aliased; See Rules for naming IBM WebSphere MQ objects . The object can be a queue or a topic as defined by **TARGTYPE**. The maximum length is 48 characters.

This parameter is supported only on alias queues.

This object needs to be defined only when an application process opens the alias queue.

This parameter is a synonym of the parameter **TARGQ**; **TARGQ** is retained for compatibility. If you specify **TARGET**, you cannot also specify **TARGQ**.

TARGTYPE(*string***)**
The type of object to which the alias resolves.

QUEUE The alias resolves to a queue.

TOPIC The alias resolves to a topic.

TRIGDATA(*string***)**
The data that is inserted in the trigger message. The maximum length of the string is 64 bytes.

This parameter is supported only on local and model queues.

For a transmission queue on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS, you can use this parameter to specify the name of the channel to be started.

This parameter can also be changed using the MQSET API call.

TRIGDPTH(*integer***)**
The number of messages that have to be on the queue before a trigger message is written, if **TRIGTYPE** is DEPTH. The value must be in the range 1 - 999,999,999.

This parameter is supported only on local and model queues.

This parameter can also be changed using the MQSET API call.

TRIGGER &**NOTRIGGER**
Specifies whether trigger messages are written to the initiation queue, named by the **INITQ** parameter, to trigger the application, named by the **PROCESS** parameter:

TRIGGER
Triggering is active, and trigger messages are written to the initiation queue.

NOTRIGGER
Triggering is not active, and trigger messages are not written to the initiation queue.

This parameter is supported only on local and model queues.

This parameter can also be changed using the MQSET API call.

TRIGMPRI(*integer***)**
The message priority number that triggers this queue. The value must be in the range zero through to the **MAXPRTY** queue manager parameter; see "DISPLAY QMGR" on page 672 for details.

This parameter can also be changed using the MQSET API call.

TRIGTYPE
Specifies whether and under what conditions a trigger message is written to the initiation queue. The initiation queue is (named by the **INITQ** parameter.

This parameter is supported only on local and model queues.

FIRST Whenever the first message of priority equal to or greater than the priority specified by the **TRIGMPRI** parameter of the queue arrives on the queue.

EVERY Every time a message arrives on the queue with priority equal to or greater than the priority specified by the **TRIGMPRI** parameter of the queue.

DEPTH When the number of messages with priority equal to or greater than the priority specified by **TRIGMPRI** is equal to the number indicated by the **TRIGDPTH** parameter.

NONE No trigger messages are written.

This parameter can also be changed using the MQSET API call.

USAGE
Queue usage.

This parameter is supported only on local and model queues.

NORMAL The queue is not a transmission queue.

XMITQ The queue is a transmission queue, which is used to hold messages that are destined for a remote queue manager. When an application puts a message to a remote queue, the message is stored on the appropriate transmission queue. It stays there, awaiting transmission to the remote queue manager.

If you specify this option, do not specify values for **CLUSTER** and **CLUSNL** and do not specify **INDXTYPE**(MSGTOKEN) or **INDXTYPE**(GROUPID).

XMITQ(*string***)**
The name of the transmission queue to be used for forwarding messages to the remote queue. **XMITQ** is used with either remote queue or queue manager alias definitions.

This parameter is supported only on remote queues.

If **XMITQ** is blank, a queue with the same name as **RQMNAME** is used as the transmission queue.

This parameter is ignored if the definition is being used as a queue manager alias and **RQMNAME** is the name of the local queue manager.

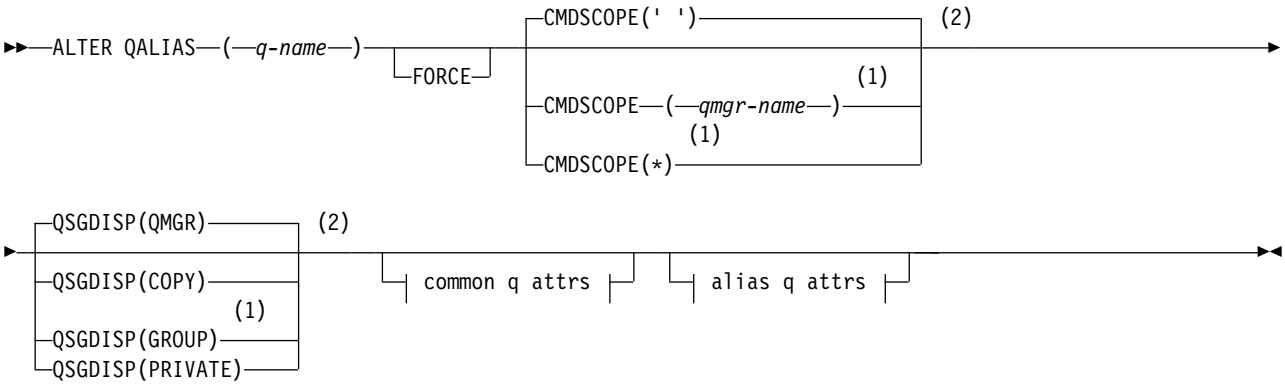
It is also ignored if the definition is used as a reply-to queue alias definition.

ALTER QALIAS:

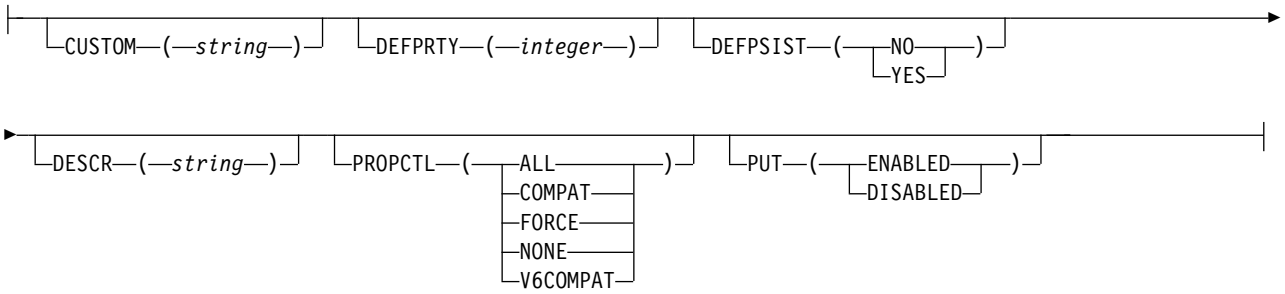
Use the MQSC command ALTER QALIAS to alter the parameters of an alias queue.

Synonym: ALT QA

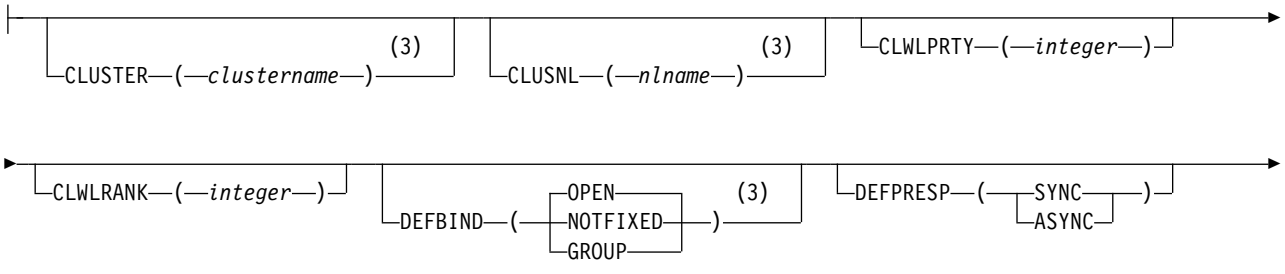
ALTER QALIAS

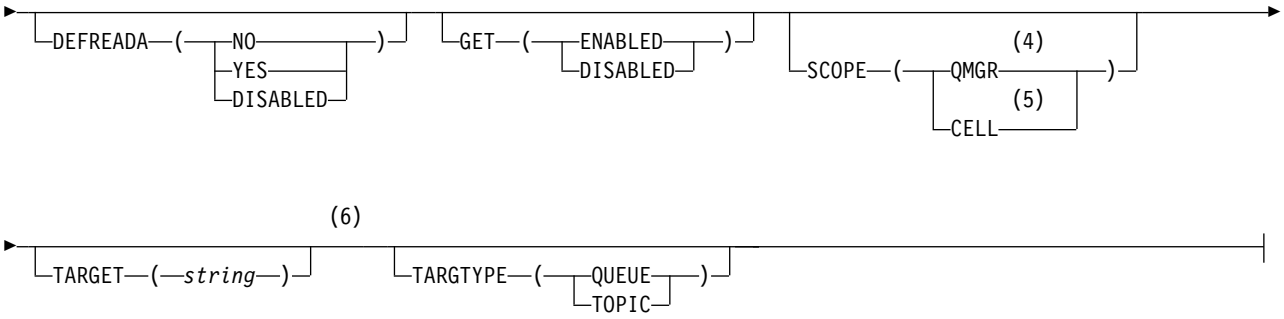


Common q attrs:



Alias q attrs:





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Valid only on AIX, HP-UX, z/OS, IBM i, Solaris, and Windows.

4 Valid only on IBM i, Windows, UNIX, and Linux systems.

5 Valid only on Windows, UNIX, and Linux systems.

6 The TARGQ parameter is available for compatibility with previous releases. It is a synonym of TARGET; you cannot specify both parameters.

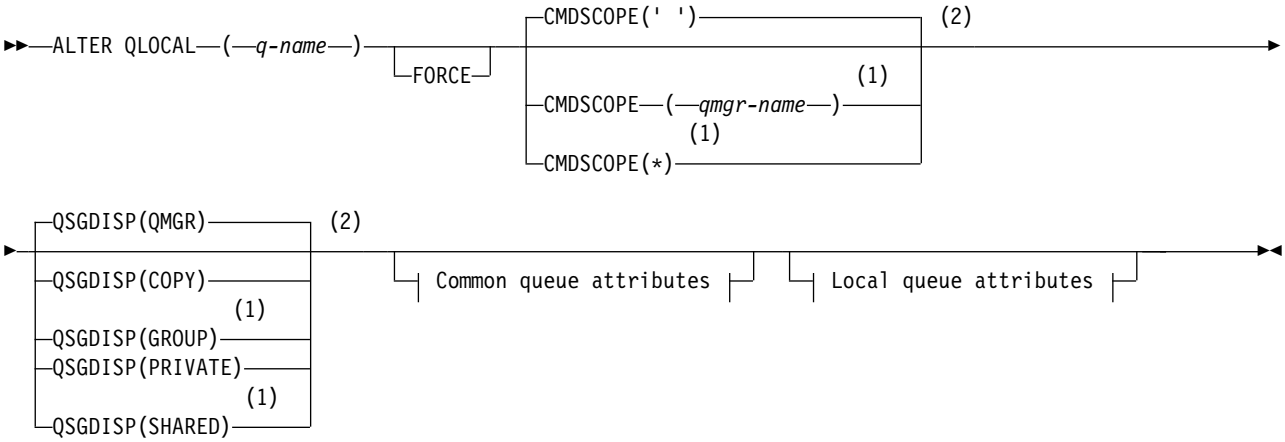
The parameters are described in "ALTER queues" on page 386.

ALTER QLOCAL:

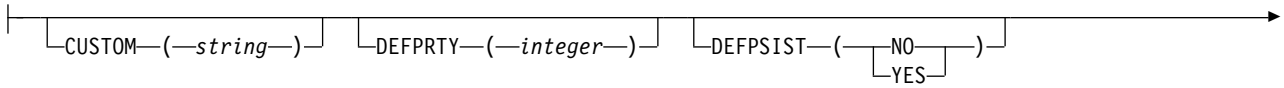
Use the MQSC command **ALTER QLOCAL** to alter the parameters of a local queue.

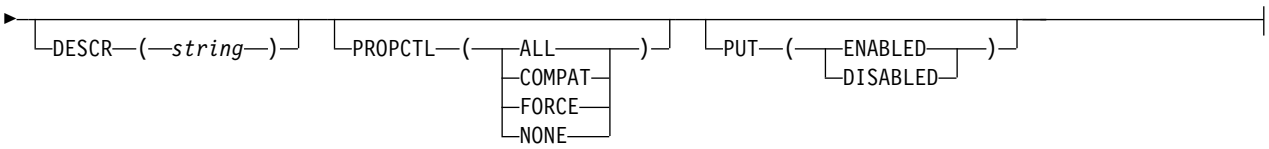
Synonym: ALT QL

ALTER QLOCAL

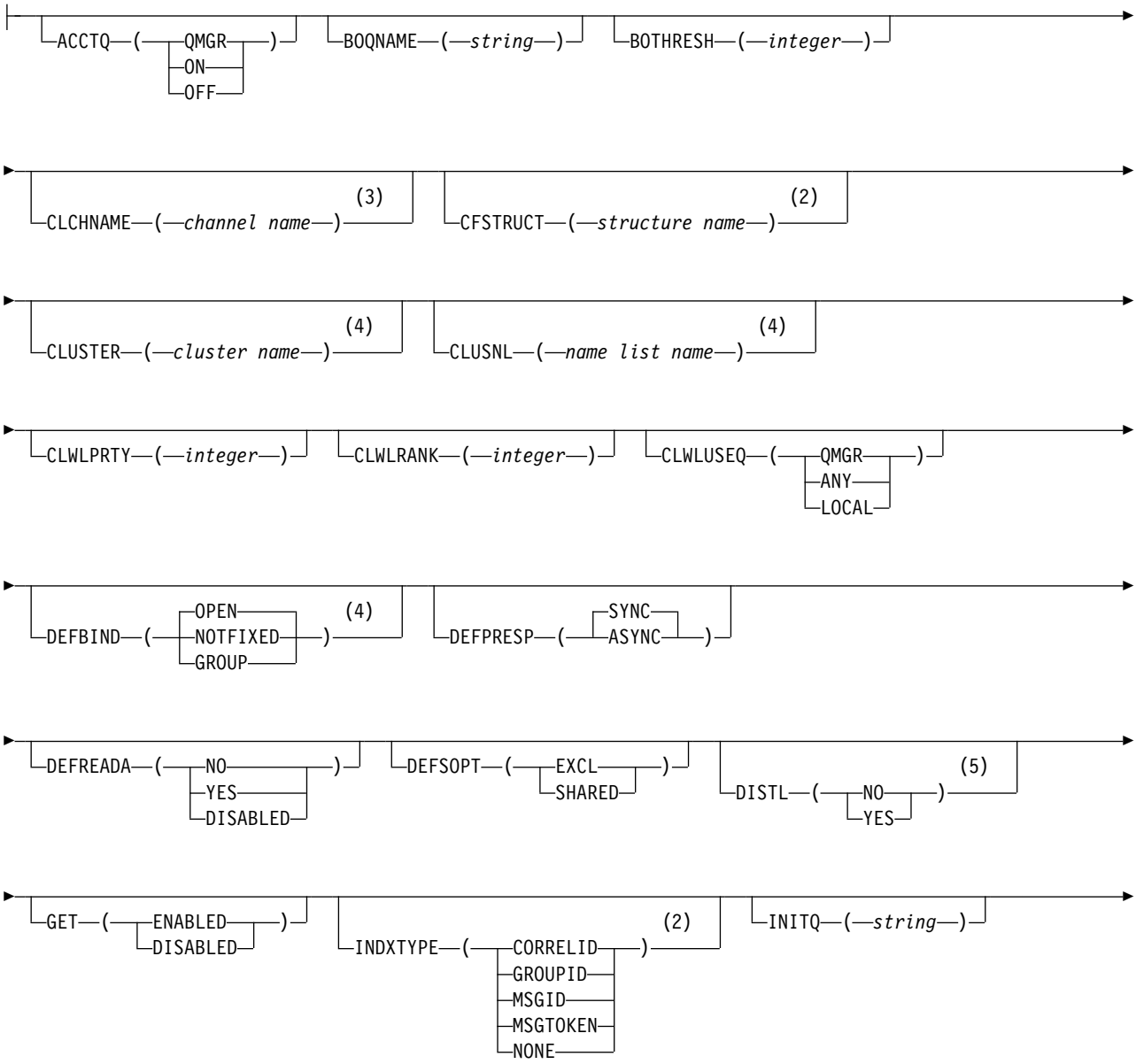


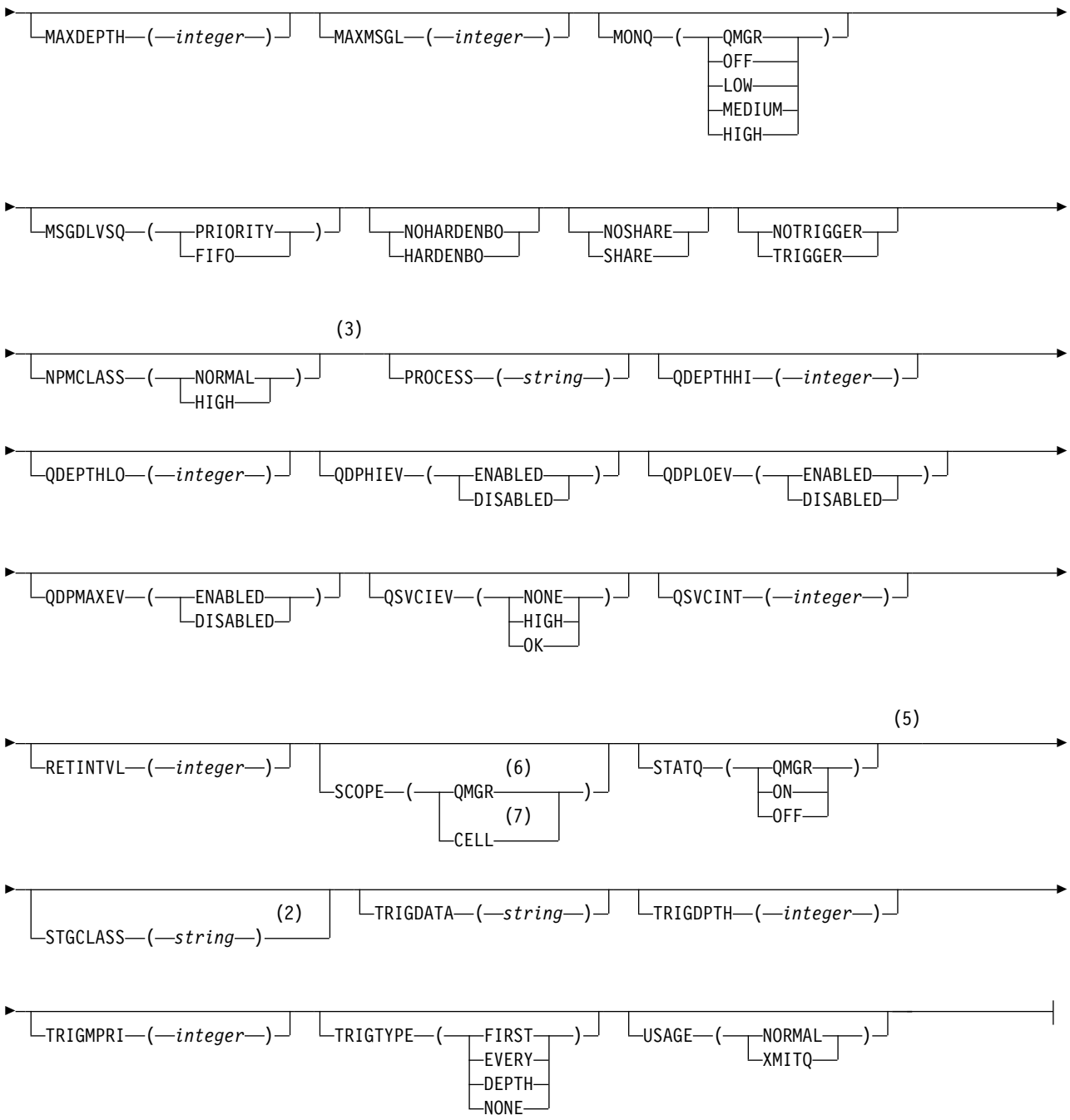
Common queue attributes:





Local queue attributes:





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

4 Valid on IBM i, UNIX, Linux,Windows, and z/OS systems.

5 Valid on IBM i, UNIX, Linux, and Windows systems.

6 Valid on IBM i, UNIX, Linux, and Windows systems.

7 Valid on UNIX, Linux, and Windows systems.

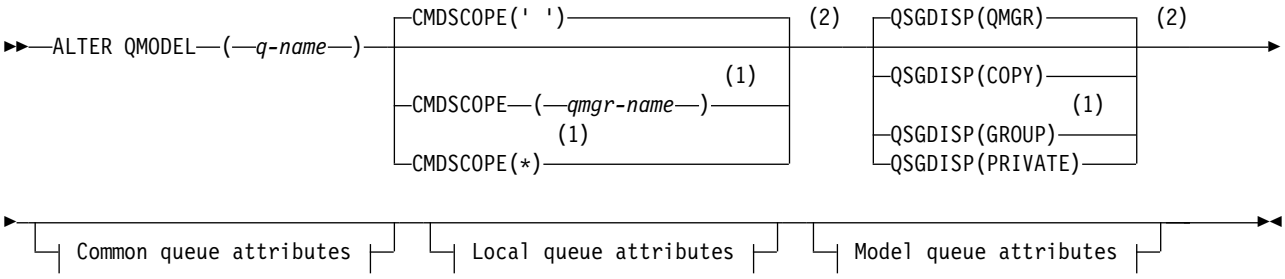
The parameters are described in "ALTER queues" on page 386.

ALTER QMODEL:

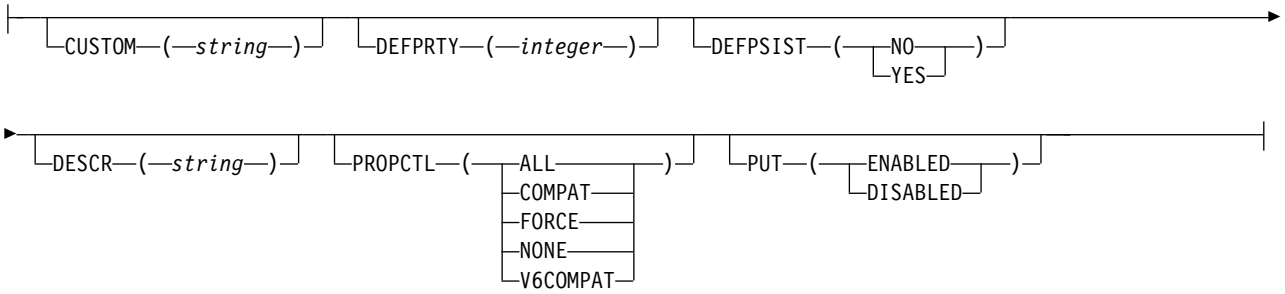
Use the MQSC command `ALTER QMODEL` to alter the parameters of a model queue.

Synonym: `ALT QM`

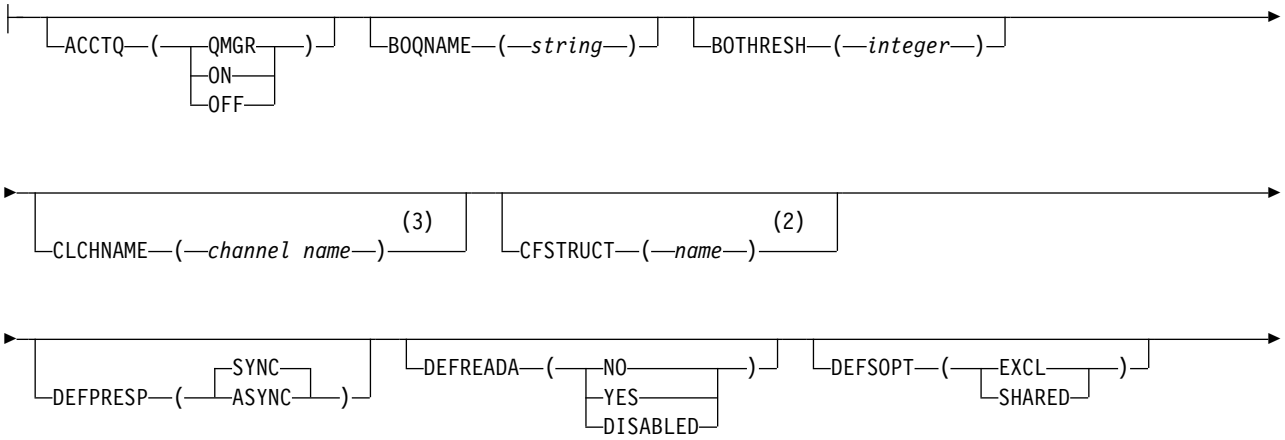
ALTER QMODEL

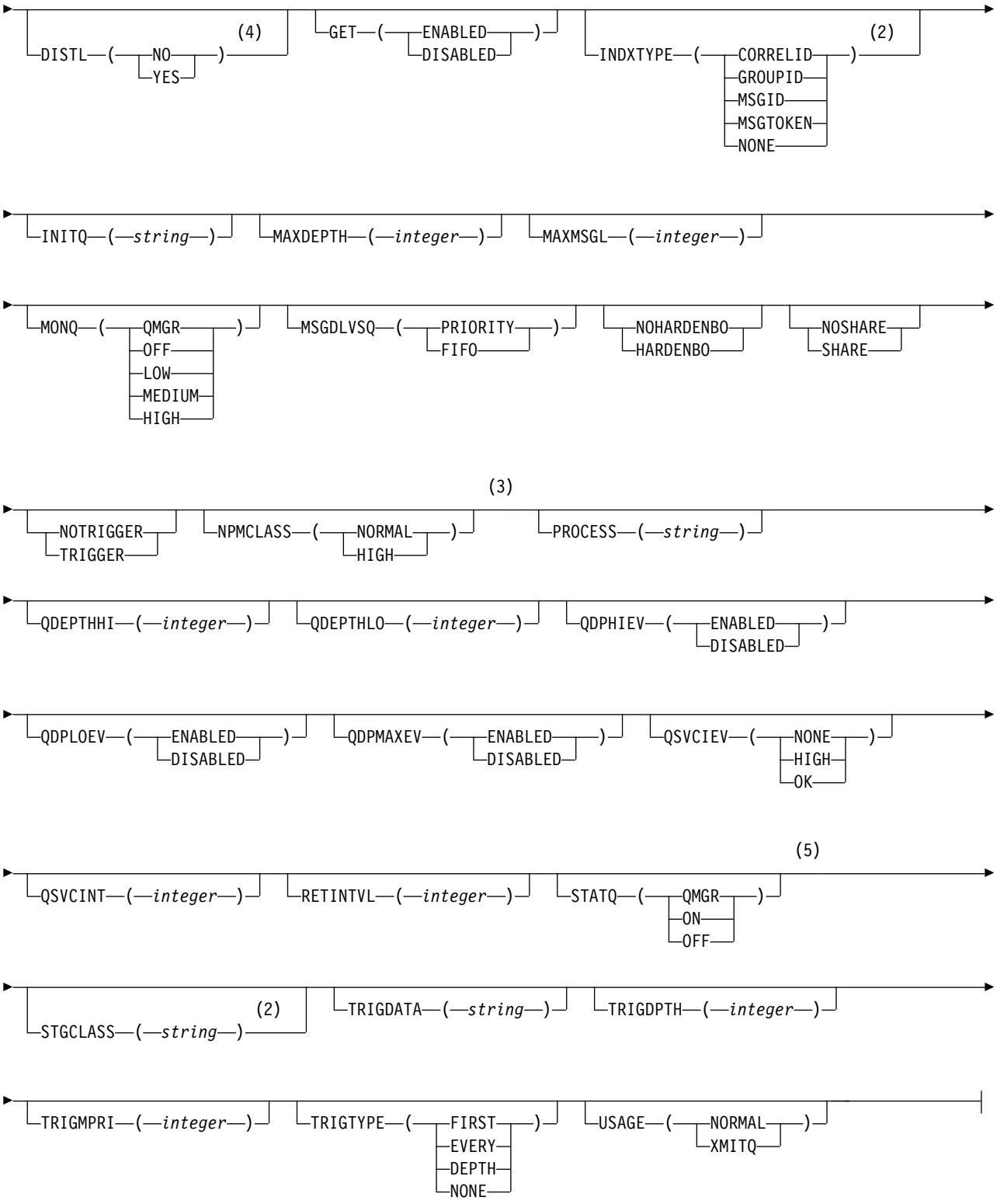


Common queue attributes:



Local queue attributes:





Model queue attributes:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

4 Valid only on AIX, HP-UX, IBM i, Solaris, and Windows.

5 Valid only on IBM i, UNIX systems, and Windows.

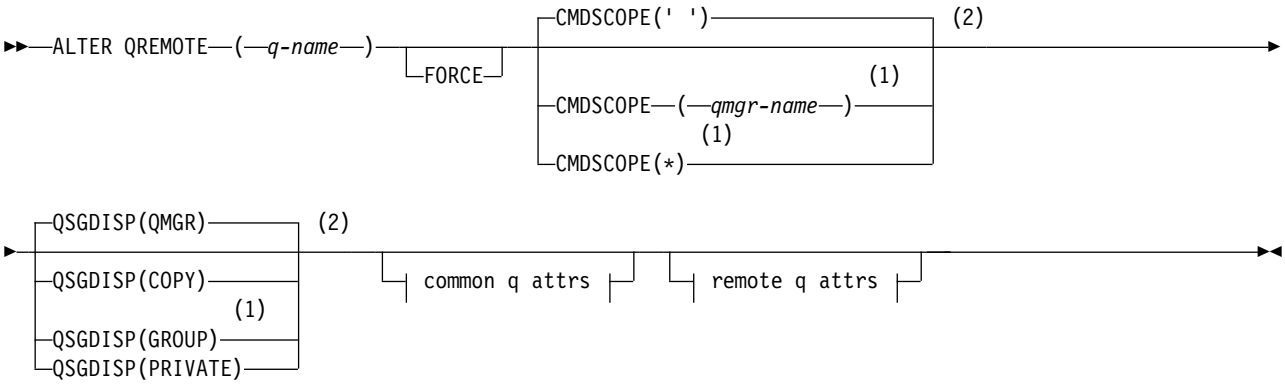
The parameters are described in "ALTER queues" on page 386.

ALTER QREMOTE:

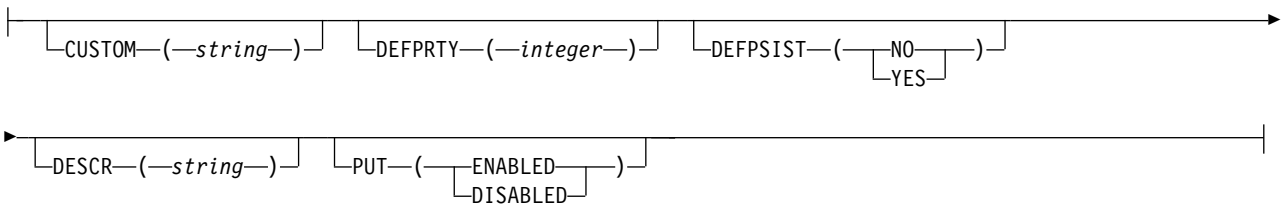
Use the MQSC command ALTER QREMOTE to alter the parameters of a local definition of a remote queue, a queue-manager alias, or a reply-to queue alias.

Synonym: ALT QR

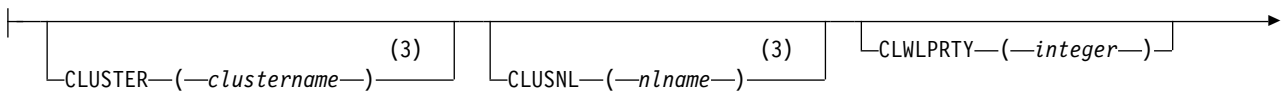
ALTER QREMOTE

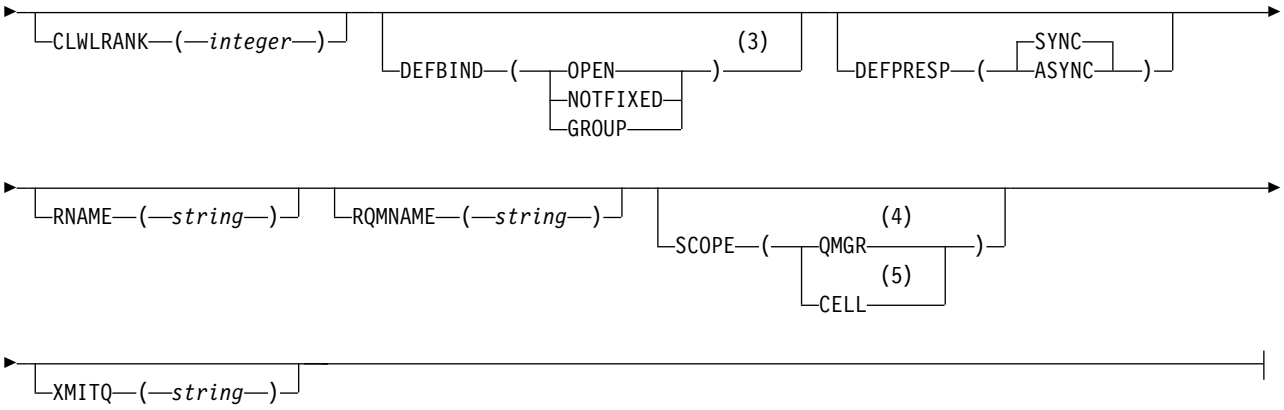


Common q attrs:



Remote q attrs:





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Valid only on AIX, HP-UX, z/OS, IBM i, Solaris, and Windows.

4 Valid only on IBM i, Windows, UNIX, and Linuxsystems.

5 Valid only on Windows, UNIX, and Linuxsystems.

The parameters are described in "ALTER queues" on page 386.

ALTER SERVICE:

Use the MQSC command ALTER SERVICE to alter the parameters of an existing WebSphere MQ service definition.

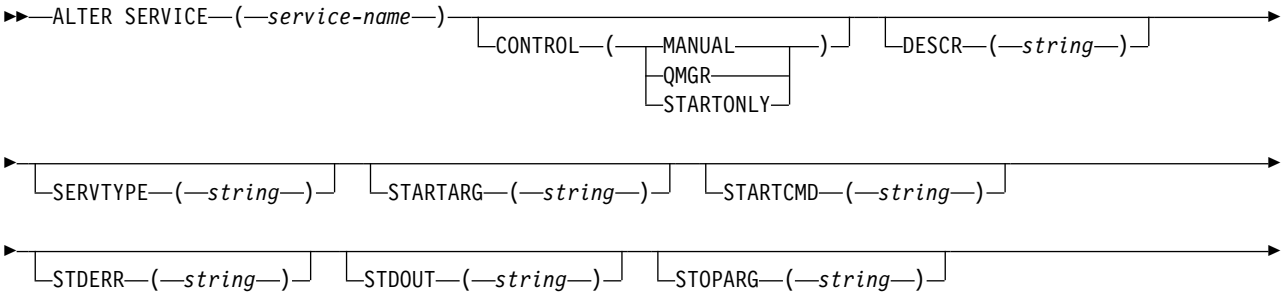
UNIX and Linux	Windows
✓	✓

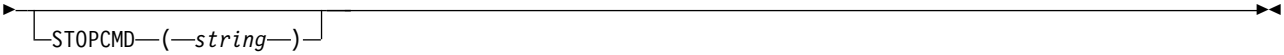
Parameters not specified in the ALTER SERVICE command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Parameter descriptions for ALTER SERVICE" on page 417

Synonym:

ALTER SERVICE





Parameter descriptions for ALTER SERVICE

The parameter descriptions apply to the ALTER SERVICE and DEFINE SERVICE commands, with the following exceptions:

- The **LIKE** parameter applies only to the DEFINE SERVICE command.
- The **NOREPLACE** and **REPLACE** parameter applies only to the DEFINE SERVICE command.

(service-name)
Name of the WebSphere MQ service definition (see Rules for naming IBM WebSphere MQ objects).

The name must not be the same as any other service definition currently defined on this queue manager (unless REPLACE is specified).

CONTROL(*string***)**
Specifies how the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the START SERVICE and STOP SERVICE commands.

QMGR
The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the service when an operator issues the DISPLAY SERVICE command (see "DISPLAY SERVICE" on page 717).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LIKE(*service-name***)**
The name of a service the parameters of which are used to model this definition.

This parameter applies only to the DEFINE SERVICE command.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from the default definition for services on this queue manager. Not completing this parameter is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.SERVICE)
```

A default service is provided but it can be altered by the installation of the default values required. See Rules for naming IBM WebSphere MQ objects .

REPLACE and NOREPLACE
Whether the existing definition is to be replaced with this one.

This parameter applies only to the DEFINE SERVICE command.

REPLACE
The definition must replace any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE
The definition should not replace any existing definition of the same name.

SERVTYPE
Specifies the mode in which the service is to run:

COMMAND
A command service object. Multiple instances of a command service object can be executed concurrently. You cannot monitor the status of command service objects.

SERVER
A server service object. Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the DISPLAY SVSTATUS command.

STARTARG(*string***)**
Specifies the arguments to be passed to the user program at queue manager startup.

STARTCMD(*string***)**
Specifies the name of the program which is to run. You must specify a fully qualified path name to the executable program.

STDERR(*string***)**
Specifies the path to a file to which the standard error (stderr) of the service program is redirected. If the file does not exist when the service program is started, the file is created. If this value is blank then any data written to stderr by the service program is discarded.

STDOUT(*string***)**
Specifies the path to a file to which the standard output (stdout) of the service program is redirected. If the file does not exist when the service program is started, the file is created. If this value is blank then any data written to stdout by the service program is discarded.

STOPARG(*string***)**
Specifies the arguments to be passed to the stop program when instructed to stop the service.

STOPCMD(*string***)**
Specifies the name of the executable program to run when the service is requested to stop. You must specify a fully qualified path name to the executable program.

Replaceable inserts can be used for any of the STARTCMD, STARTARG, STOPCMD, STOPARG, STDOUT or STDERR strings, for more information, see Replaceable inserts on service definitions.

Related information:

Working with services

ALTER SUB:

Use the MQSC command ALTER SUB to alter the characteristics of an existing subscription.

UNIX and Linux	Windows
✓	✓

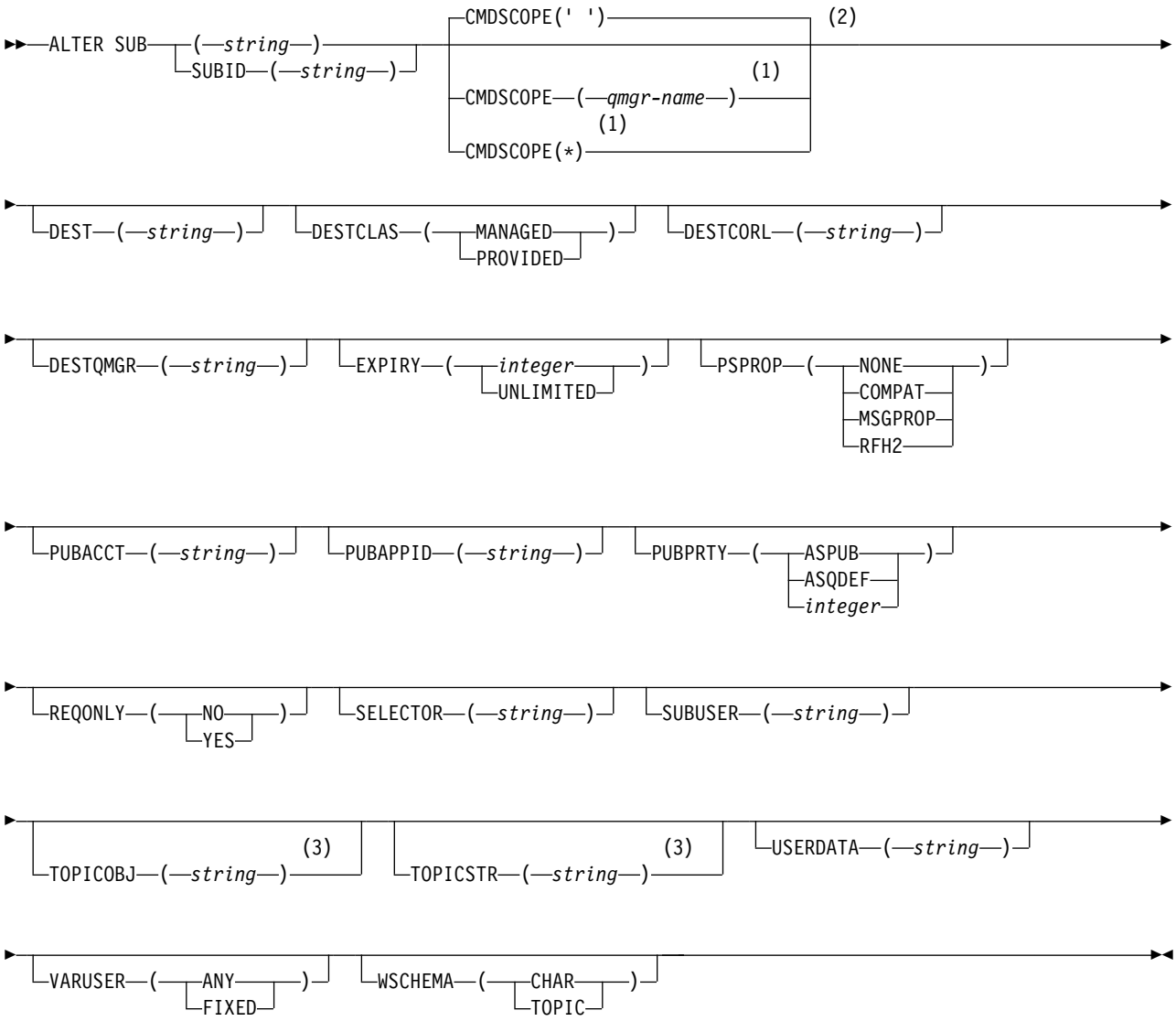
Parameters not specified in the ALTER SUB command result in the existing values for those parameters being left unchanged.

- Syntax diagram

- "Usage notes for ALTER SUB"
- "Parameter descriptions for ALTER SUB" on page 420

Synonym: ALT SUB

ALTER SUB



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 At least one of **TOPICSTR** and **TOPICOBJ** must be present on **DEFINE**.

Usage notes for ALTER SUB

1. The following are valid forms of the command:

```
ALT SUB(xyz)
ALT SUB SUBID(123)
ALT SUB(xyz) SUBID(123)
```

2. Although permitted on the command, you cannot alter the following fields using DEF SUB (REPLACE) or ALTER SUB:
   - TOPICOBJ
   - TOPICSTR
   - WSCHEMA
   - SELECTOR
   - SUBSCOPE
   - DESTCLAS
3. At the time the ALT SUB command processes, no check is performed that the named DEST or DESTQMGR exists. These names are used at publishing time as the *ObjectName* and *ObjectQMgrName* for an MQOPEN call. These names are resolved according to the WebSphere MQ name resolution rules.

Parameter descriptions for ALTER SUB

(string)
A mandatory parameter. Specifies the unique name for this subscription, see **SUBNAME** property.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is processed when the queue manager is a member of a queue-sharing group.

'' The command is processed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is processed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of setting this value is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

DEST(*string***)**
The destination for messages published to this subscription; this parameter is the name of a queue.

DESTCORL(*string***)**
The *CorrelId* used for messages published to this subscription.

DESTQMGR(*string***)**
The destination queue manager for messages published to this subscription. You must define the channels to the remote queue manager, for example, the XMITQ, and a sender channel. If you do not, messages do not arrive at the destination.

EXPIRY
The time to expiry of the subscription object from the creation date and time.

(*integer***)**
The time to expiry, in tenths of a second, from the creation date and time.

UNLIMITED
There is no expiry time. This is the default option supplied with the product.

PSPROP
The manner in which publish subscribe related message properties are added to messages sent to this subscription.

NONE
Do not add publish subscribe properties to the message.

COMPAT
Publish subscribe properties are added within an MQRFH version 1 header unless the message was published in PCF format.

MSGPROP
Publish subscribe properties are added as message properties.

RFH2 Publish subscribe properties are added within an MQRFH version 2 header.

PUBACCT(*string***)**
Accounting token passed by the subscriber, for propagation into messages published to this subscription in the *AccountingToken* field of the MQMD.

PUBAPPID(*string***)**
Identity data passed by the subscriber, for propagation into messages published to this subscription in the *ApplIdentityData* field of the MQMD.

PUBPRTY
The priority of the message sent to this subscription.

ASPUB
Priority of the message sent to this subscription is taken from the priority supplied in the published message.

ASQDEF
Priority of the message sent to this subscription is taken from the default priority of the queue defined as a destination.

(integer)
An integer providing an explicit priority for messages published to this subscription.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.

NO All publications on the topic are delivered to this subscription.

YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

This parameter is equivalent to the subscribe option MQSO_PUBLICATIONS_ON_REQUEST.

SUBLEVEL(*integer***)**
The level within the subscription hierarchy at which this subscription is made. The range is zero through 9.

SUBUSER(*string***)**
Specifies the user ID that is used for security checks that are performed to ensure that publications can be put to the destination queue associated with the subscription. This ID is either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription. The length of this parameter must not exceed 12 characters.

USERDATA(*string***)**
Specifies the user data associated with the subscription. The string is a variable length value that can be retrieved by the application on an MQSUB API call and passed in a message sent to this subscription as a message property.

V 7.5.0.8 From Version 7.5.0, Fix Pack 8, an IBM WebSphere MQ classes for JMS application can retrieve the subscription user data from the message by using the constant JMS_IBM_SUBSCRIPTION_USER_DATA in the JmsConstants interface with the method javax.jms.Message.getStringProperty(java.lang.String). For more information, see Retrieval of user subscription data.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.

ANY Any user can connect to and takeover ownership of the subscription.

FIXED
Takeover by another **USERID** is not permitted.

ALTER TOPIC:

Use ALTER TOPIC to alter the parameters of an existing IBM WebSphere MQ topic object.

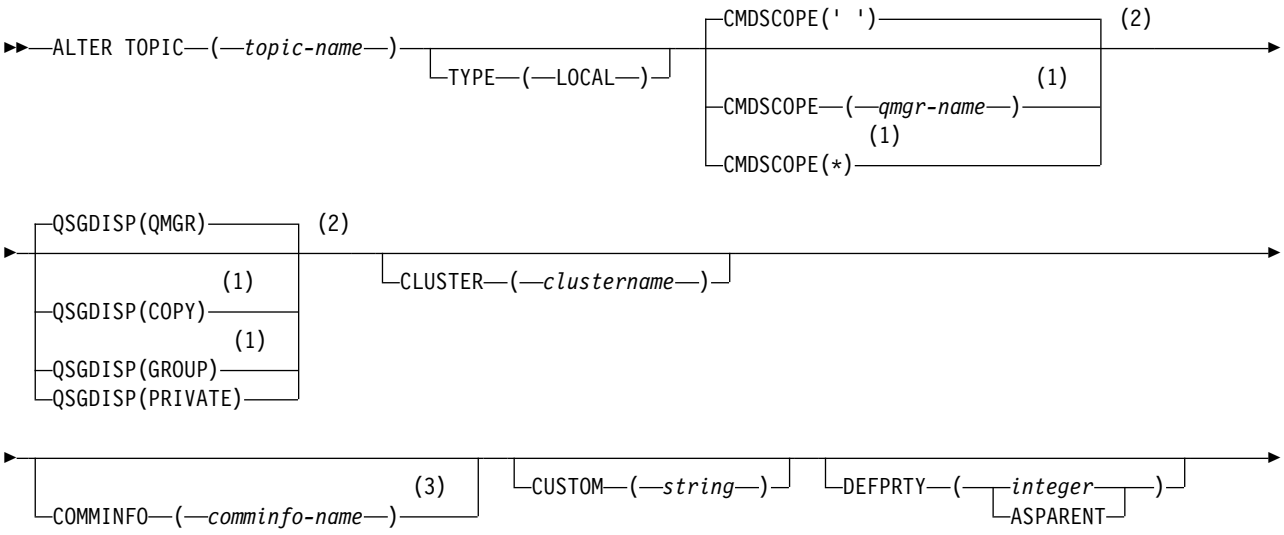
UNIX and Linux	Windows
✓	✓

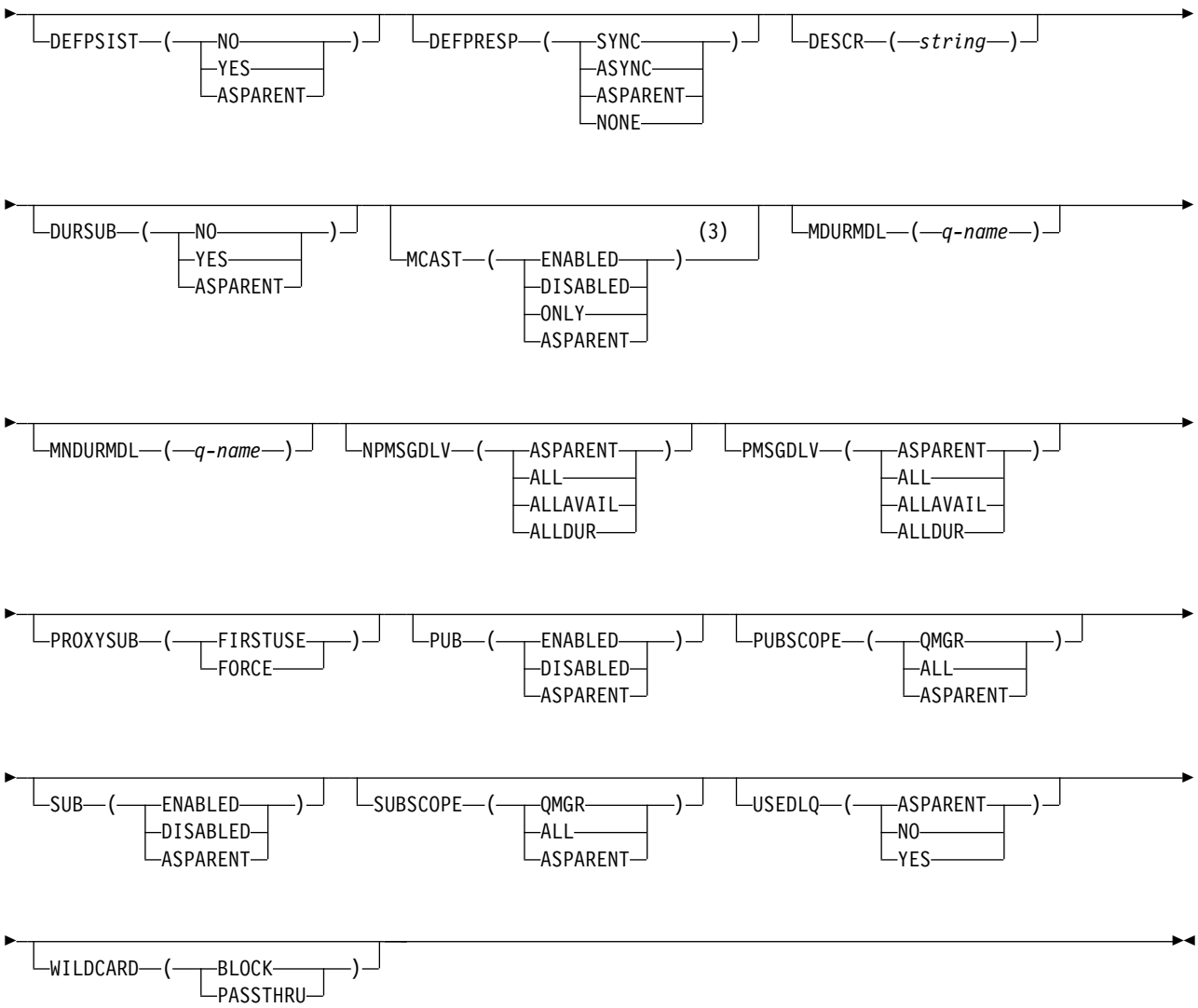
Parameters not specified in the ALTER TOPIC command result in the existing values for those parameters being left unchanged.

- Syntax diagram
- "Parameter descriptions for ALTER TOPIC" on page 423

Synonym: ALT TOPIC

ALTER TOPIC





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Parameter descriptions for ALTER TOPIC

(topic-name)

Name of the WebSphere MQ topic definition (see Rules for naming IBM WebSphere MQ objects). The maximum length is 48 characters.

The name must not be the same as any other topic definition currently defined on this queue manager (unless REPLACE is specified).

CLUSTER

The name of the cluster to which this topic belongs.

' ' This topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers.

string The topic belongs to this cluster.

Leave this parameter blank on the system topics SYSTEM.BASE.TOPIC and SYSTEM.DEFAULT.TOPIC, except in special circumstances to do with migration, documented elsewhere.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

COMMINFO(*comminfo-name*)
The name of the communication information object associated with this topic object.

CUSTOM(*string*)
The custom attribute for new features.

This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form `NAME(VALUE)`. Single quotes must be escaped with another single quote.

This description will be updated when features using this attribute are introduced. At the moment there are no possible values for *Custom*.

DEFPRTY(*integer*)
The default priority of messages published to the topic.

(*integer*)
The value must be in the range zero (the lowest priority), through to the MAXPRTY queue manager parameter (MAXPRTY is 9).

ASPARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DEFPSIST
Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_TOPIC_DEF option.

ASPARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

NO Messages on this queue are lost during a restart of the queue manager.

YES Messages on this queue survive a restart of the queue manager.

On z/OS, N and Y are accepted as synonyms of NO and YES.

DEFPRESP
Specifies the put response to be used when applications specify the MQPMO_RESPONSE_AS_DEF option.

ASPARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

SYNC Put operations to the queue that specify MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_SYNC_RESPONSE had been specified instead. Fields in the MQMD and MQPMO are returned by the queue manager to the application.

ASYNC
Put operations to the queue that specify MQPMO_RESPONSE_AS_Q_DEF are always issued as if MQPMO_ASYNC_RESPONSE had been specified instead. Some fields in the MQMD and MQPMO are not returned by the queue manager to the application. However, an improvement in performance might be seen for messages put in a transaction and any non-persistent messages

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY TOPIC command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

DURSUB
Specifies whether applications are permitted to make durable subscriptions on this topic.

ASPARENT
Whether durable subscriptions can be made on this topic is based on the setting of the closest parent administrative topic object in the topic tree.

NO Durable subscriptions cannot be made on this topic.

YES Durable subscriptions can be made on this topic.

MCAST
Specifies whether multicast is allowable in the topic tree. The values are:

ASPARENT
The multicast attribute of the topic is inherited from the parent.

DISABLED
No multicast traffic is allowed at this node.

ENABLED
Multicast traffic is allowed at this node.

ONLY Only subscriptions from a multicast capable client are allowed.

MDURMDL(*string***)**
The name of the model queue to be used for durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming IBM WebSphere MQ objects). The maximum length is 48 characters.

If MDURMDL is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for MDURMDL.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.DURABLE

MNDURMDL(*string***)**
The name of the model queue to be used for non-durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming IBM WebSphere MQ objects). The maximum length is 48 characters.

If MNDURMDL is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for MNDURMDL.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.NDURABLE.

NPMSGDLV
The delivery mechanism for non-persistent messages published to this topic:

ASPARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

ALL Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

ALLAVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ALLDUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT calls fails.

PMSGDLV
The delivery mechanism for persistent messages published to this topic:

ASPARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

ALL Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

ALLAVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ALLDUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT calls fails.

PROXYSUB
Controls when a proxy subscription is sent for this topic, or topic strings below this topic, to neighboring queue managers when in a publish/subscribe cluster or hierarchy. For more details, see More on routing mechanisms.

FIRSTUSE
For each unique topic string at or below this topic object, a proxy subscription is asynchronously sent to all neighboring queue managers when a local subscription is created or a proxy subscription is received that is propagated to further directly connected queue managers in a hierarchy.

FORCE
A wildcard proxy subscription that matches all topic strings at and below this point in the topic tree is sent to neighboring queue managers even if no local subscriptions exist.

Note: The proxy subscription is sent when this value is set on DEFINE or ALTER. When set on a clustered topic, all queue managers in the cluster issue the wildcard proxy subscription to all other queue managers in the cluster.

PUB Controls whether messages can be published to this topic.

ASPARENT
Whether messages can be published to the topic is based on the setting of the closest parent administrative topic object in the topic tree.

ENABLED
Messages can be published to the topic (by suitably authorized applications).

DISABLED
Messages cannot be published to the topic.

PUBSCOPE
Determines whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster.

Note: You can restrict the behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message options.

ASPARENT
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree that relates to this topic.

QMGR
Publications for this topic are not propagated to connected queue managers.

ALL Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

QSGDISP
This parameter applies to z/OS only.

Specifies the disposition of the object within the group.

QSGDISP	ALTER
COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

QSGDISP	ALTER

GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero:
	`DEFINE TOPIC(name)`
	`REPLACE QSGDISP(COPY)`
	
	The ALTER for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with QSGDISP(QMGR) or QSGDISP(COPY). Any object residing in the shared repository is unaffected.
QMGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

SUB Controls whether applications are to be permitted to subscribe to this topic.

ASPARENT
Whether applications can subscribe to the topic is based on the setting of the closest parent administrative topic object in the topic tree.

ENABLED
Subscriptions can be made to the topic (by suitably authorized applications).

DISABLED
Applications cannot subscribe to the topic.

SUBSCOPE
Determines whether this queue manager subscribes to publications in this queue manager or in the network of connected queue managers. If subscribing to all queue managers, the queue manager propagates subscriptions to them as part of a hierarchy or as part of a publish/subscribe cluster.

Note: You can restrict the behavior on a subscription-by-subscription basis, using **MQPMO_SCOPE_QMGR** on the Subscription Descriptor or **SUBSCOPE(QMGR)** on **DEFINE SUB**. Individual subscribers can override the **SUBSCOPE** setting of ALL by specifying the **MQSO_SCOPE_QMGR** subscription option when creating a subscription.

ASPARENT
Whether this queue manager subscribes to publications in the same way as the setting of the first parent administrative node found in the topic tree relating to this topic.

QMGR
Only publications that are published on this queue manager reach the subscriber.

ALL A publication made on this queue manager or on another queue manager reaches the subscriber. Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

TOPICSTR(*string***)**
The topic string represented by this topic object definition. This parameter is required and cannot contain the empty string.

The topic string must not be the same as any other topic string already represented by a topic object definition.

The maximum length of the string is 10,240 characters.

TYPE (topic-type)
If this parameter is used it must follow immediately after the *topic-name* parameter on all platforms except z/OS.

LOCAL
A local topic object.

USEDLQ
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue.

ASPARENT
Determines whether to use the dead-letter queue using the setting of the closest administrative topic object in the topic tree.

NO Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message. The MQPUT of an application to a topic fails in accordance with the settings of NPMSGDLV and PMSGDLV.

YES When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used. If the queue manager does not provide the name of a dead-letter queue, then the behavior is as for NO.

WILDCARD
The behavior of wildcard subscriptions with respect to this topic.

PASSTHRU
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic.

BLOCK
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

The value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This scenario applies also if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the WILDCARD attribute is created using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

CLEAR QLOCAL:

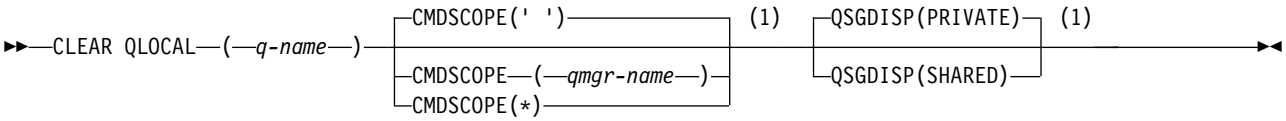
Use the MQSC command CLEAR QLOCAL to clear the messages from a local queue.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for CLEAR QLOCAL" on page 430

Synonym: CLEAR QL

CLEAR QLOCAL



Notes:

1 Valid only on z/OS.

Parameter descriptions for CLEAR QLOCAL

You must specify which local queue you want to clear.

The command fails if either:

- The queue has uncommitted messages that have been put on the queue under syncpoint
- The queue is currently open by an application (with any open options)

If an application has this queue open, or has a queue open that eventually resolves to this queue, the command fails. The command also fails if this queue is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open.

(q-name)
The name of the local queue to be cleared. The name must be defined to the local queue manager.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to SHARED.

' '
The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*
The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP
Specifies whether the queue definition is shared. This parameter applies to z/OS only.

PRIVATE
Clear only the private queue named *q-name*. The queue is private if it was defined using a command that had the parameters QSGDISP(COPY) or QSGDISP(QMGR). This is the default value.

SHARED
Clear only the shared queue named *q-name*. The queue is shared if it was defined using a command that had the parameters QSGDISP(SHARED).

CLEAR TOPICSTR:

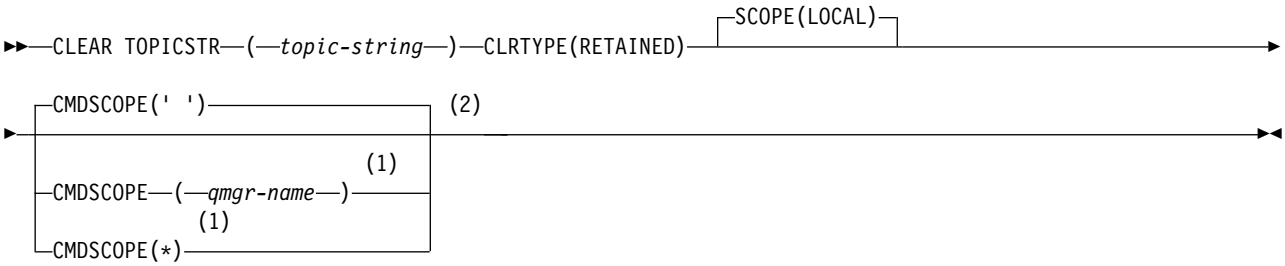
Use the MQSC command CLEAR TOPICSTR to clear the retained message which is stored for the specified topic string.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- Usage notes for CLEAR TOPICSTR
- Parameter descriptions for CLEAR TOPICSTR

Synonym: None.

CLEAR TOPICSTR



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for CLEAR TOPICSTR

1. If the topic string specified has no retained message the command will complete successfully. You can find out whether a topic string has a retained message by using the DISPLAY TPSTATUS command. The RETAINED field shows whether there is a retained message.
2. The topic-string input parameter on this command must match the topic you want to act on. You are advised to keep the character strings in your topic strings as characters that can be used from location issuing the command. If you issue commands using MQSC, you will have fewer characters available to you than if you are using an application submitting PCF messages, such as the WebSphere MQ Explorer.

Parameter descriptions for CLEAR TOPICSTR

You must specify which topic string you want to remove the retained publication from.

(topic-string)
The topic string to be cleared. This string can represent several topics to be cleared by using wildcards as shown in the following table:

Special Character	Behavior
#	Wildcard, multiple topic level
+	Wildcard, single topic level
Note: the '+' and '#' are not treated as wildcards if they are mixed in with other characters (including themselves) within a topic level. In the following string, the '#' and '+' characters are treated as ordinary characters. `level0/level1/#+/level3/level#`	

To illustrate the effect of wildcards, the following example is used.

Clearing the following topic:

```
/a/b/#/z
```

clears the following topics:

```
/a/b/z
/a/b/c/z
/a/b/c/y/z
```

CLRTYPE

This is a mandatory parameter.

The value must be:

RETAINED
Remove the retained publication from the specified topic string.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the name of the local queue manager, if the shared queue object definition has its queue-sharing group disposition attribute QSGDISP set to SHARED.

' '
The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

*****
The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

SCOPE

The scope of the deletion of retained messages.

The value can be:

LOCAL
The retained message is removed from the specified topic string at the local queue manager only. This is the default value.

DEFINE AUTHINFO:

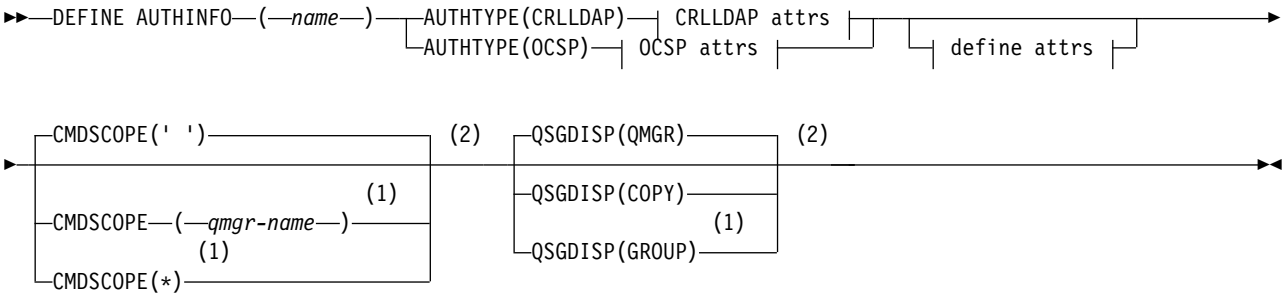
Use the MQSC command DEFINE AUTHINFO to define an authentication information object. These objects contain the definitions required to perform certificate revocation checking using OCSP or Certificate Revocation Lists (CRLs) on LDAP servers.

UNIX and Linux	Windows
✓	✓

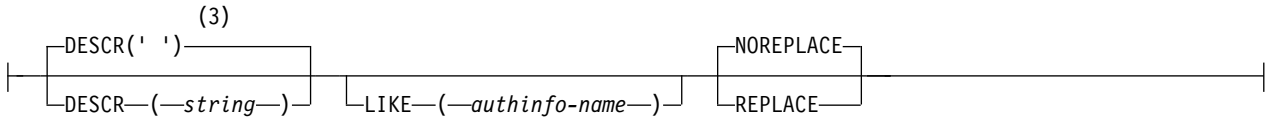
- Syntax diagram
- "Usage Notes for DEFINE AUTHINFO" on page 434
- "Parameter descriptions for DEFINE AUTHINFO" on page 434

Synonym: DEF AUTHINFO

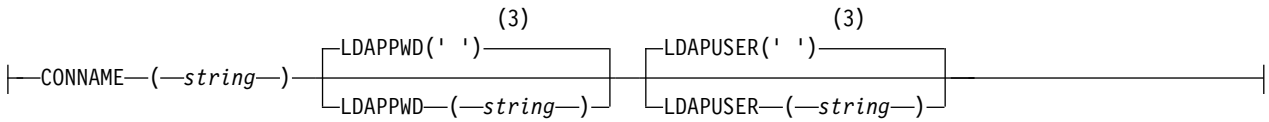
DEFINE AUTHINFO



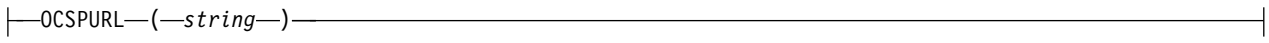
Define attrs:



CRLLDAP attrs:



OCSP attrs:



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on WebSphere MQ for z/OS.

2 Valid only on z/OS.

3 This command is the default supplied with WebSphere MQ, but your installation might have changed it.

Usage Notes for DEFINE AUTHINFO

On IBM i, authentication information objects are only used for channels of type CLNTCONN through use of the AMQCLCHL.TAB. Certificates are defined by Digital Certificate Manager for each certificate authority, and are verified against the LDAP servers.

Parameter descriptions for DEFINE AUTHINFO

name Name of the authentication information object. This parameter is required.

The name must not be the same as any other authentication information object name currently defined on this queue manager (unless REPLACE or ALTER is specified). See Rules for naming IBM WebSphere MQ objects.

AUTHTYPE
The type of authentication information.

CRLLDAP
Certificate Revocation List checking is done using LDAP servers.

OCSP
Certificate revocation checking is done using OCSP.

An authentication information object with AUTHTYPE(OCSP) does not apply for use on IBM i or z/OS queue managers. However, it can be specified on those platforms to be copied to the client channel definition table (CCDT) for client use.

This parameter is required.

You cannot define an authentication information object as LIKE one with a different AUTHTYPE. You cannot alter the AUTHTYPE of an authentication information object after you have created it.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

CONNAME(*string*)
The host name, IPv4 dotted decimal address, or IPv6 hexadecimal notation of the host on which the LDAP server is running, with an optional port number.

This parameter is valid only for AUTHTYPE(CRLLDAP), when it is mandatory.

If you specify the connection name as an IPv6 address, only systems with an IPv6 stack are able to resolve this address. If the AUTHINFO object is part of the CRL namelist of the queue manager, ensure that any clients using the client channel table generated by the queue manager can resolve the connection name.

On z/OS, if a CONNAME is to resolve to an IPv6 network address, a level of z/OS that supports IPv6 for connection to an LDAP server is required.

The syntax for CONNAME is the same as for channels. For example,

```
conname('hostname(nnn)')
```

where *nnn* is the port number.

The maximum length for the field is 264 characters on IBM i, UNIX systems, and Windows, and 48 characters on z/OS.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the authentication information object when an operator issues the DISPLAY AUTHINFO command (see "DISPLAY AUTHINFO" on page 578).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LDAPPWD(*string***)**
The password associated with the Distinguished Name of the user who is accessing the LDAP server. Its maximum size is 32 characters.

This parameter is valid only for AUTHTYPE(CRLLDAP).

On z/OS, the LDAPPWD used for accessing the LDAP server might not be the one defined in the AUTHINFO object. If more than one AUTHINFO object is placed in the namelist referred to by the QMGR parameter SSLCRLNL, the LDAPPWD in the first AUTHINFO object is used for accessing all LDAP Servers.

LDAPUSER(*string***)**
The Distinguished Name of the user who is accessing the LDAP server. (See the SSLPEER parameter for more information about distinguished names.)

This parameter is valid only for AUTHTYPE(CRLLDAP).

The maximum size for the user name is 1024 characters on IBM i, UNIX systems, and Windows, and 256 characters on z/OS.

On z/OS, the LDAPUSER used for accessing the LDAP Server might not be the one defined in the AUTHINFO object. If more than one AUTHINFO object is placed in the namelist referred to by the QMGR parameter SSLCRLNL, the LDAPUSER in the first AUTHINFO object is used for accessing all LDAP Servers.

On IBM i, UNIX systems, and Windows, the maximum accepted line length is defined to be BUFSIZ, which can be found in stdio.h.

LIKE(*authinfo-name***)**
The name of an authentication information object, with parameters that are used to model this definition.

On z/OS, the queue manager searches for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified. However, the group object defined is used as a LIKE object.

OCSPURL

The URL of the OCSP responder used to check for certificate revocation. This value must be an HTTP URL containing the host name and port number of the OCSP responder. If the OCSP responder is using port 80, which is the default for HTTP, then the port number can be omitted. HTTP URLs are defined in RFC 1738.

This field is case sensitive. It must start with the string http:// in lowercase. The rest of the URL might be case sensitive, depending on the OCSP server implementation. To preserve case, use single quotation marks to specify the OCSPURL parameter value, for example:

```
OCSPURL('http://ocsp.example.ibm.com')
```

This parameter is applicable only for AUTHTYPE(OCSP), when it is mandatory.

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the 'LIKE' object.

GROUP	The object definition resides in the shared repository. GROUP is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to make or refresh local copies on page set zero:
	`DEFINE AUTHINFO(name)`
	`REPLACE QSGDISP(COPY)`
	
	The DEFINE for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE

Whether the existing definition (and on z/OS, with the same disposition) is to be replaced with this one. This parameter is optional. Any object with a different disposition is not changed.

REPLACE

The definition must replace any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE

The definition must not replace any existing definition of the same name.

DEFINE CHANNEL:

Use the MQSC command **DEFINE CHANNEL** to define a new channel, and set its parameters.

UNIX and Linux	Windows
✓	✓

Synonym: DEF CHL

- "Usage notes"
- "Parameter descriptions for DEFINE CHANNEL"

Usage notes

For CLUSSDR channels, you can specify the REPLACE option only for manually created channels.

Parameter descriptions for DEFINE CHANNEL

The following table shows the parameters that are relevant for each type of channel. There is a description of each parameter after the table. Parameters are optional unless the description states that they are required.

SDR "Sender channel" on page 471

SVR "Server channel" on page 473

RCVR "Receiver channel" on page 476

RQSTR "Requester channel" on page 479

CLNTCONN
"Client-connection channel" on page 481

SVRCONN
"Server-connection channel" on page 483

CLUSSDR
"Cluster-sender channel" on page 484

CLUSRCVR
"Cluster-receiver channel" on page 487

MQTT "DEFINE CHANNEL (MQTT)" on page 489

Table 71. DEFINE and ALTER CHANNEL parameters

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	MQTT
AFFINITY					✓				
BACKLOG									✓
BATCHHB	✓	✓					✓	✓	
BATCHINT	✓	✓					✓	✓	
BATCHLIM	✓	✓					✓	✓	
BATCHSZ	✓	✓	✓	✓			✓	✓	

Table 71. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	MQTT
channel-name	✓	✓	✓	✓	✓	✓	✓	✓	✓
CHLTYPE	✓	✓	✓	✓	✓	✓	✓	✓	✓
CLNTWGHT					✓				
CLUSNL							✓	✓	
CLUSTER							✓	✓	
CLWLPRTY							✓	✓	
CLWLRANK							✓	✓	
CLWLWGHT							✓	✓	
CMDSCOPE	✓	✓	✓	✓	✓	✓	✓	✓	
COMPHDR	✓	✓	✓	✓	✓	✓	✓	✓	
COMPMSG	✓	✓	✓	✓	✓	✓	✓	✓	
CONNAME	✓	✓		✓	✓		✓	✓	
CONVERT	✓	✓					✓	✓	
DEFCDISP	✓	✓	✓	✓		✓			
DEFRECON					✓				
DESCR	✓	✓	✓	✓	✓	✓	✓	✓	✓
DISCINT	✓	✓				✓	✓	✓	
HBINT	✓	✓	✓	✓	✓	✓	✓	✓	
JAASCFG									✓
KAINT	✓	✓	✓	✓	✓	✓	✓	✓	
LIKE	✓	✓	✓	✓	✓	✓	✓	✓	✓
LOCLADDR	✓	✓		✓	✓		✓	✓	✓
LONGRTY	✓	✓					✓	✓	
LONGTMR	✓	✓					✓	✓	

Table 71. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	MQTT
MAXINST						✓			
MAXINSTC						✓			
MAXMSGL	✓	✓	✓	✓	✓	✓	✓	✓	
MCANAME	✓	✓		✓			✓	✓	
MCATYPE	✓	✓		✓			✓	✓	
MCAUSER			✓	✓		✓		✓	✓
MODENAME	✓	✓		✓	✓		✓	✓	
MONCHL	✓	✓	✓	✓		✓	✓	✓	
MRDATA			✓	✓				✓	
MREXIT			✓	✓				✓	
MRRTY			✓	✓				✓	
MRTMR			✓	✓				✓	
MSGDATA	✓	✓	✓	✓			✓	✓	
MSGEXIT	✓	✓	✓	✓			✓	✓	
NETPRTY								✓	
NPMSPEED	✓	✓	✓	✓			✓	✓	
PASSWORD	✓	✓		✓	✓		✓	✓	
PORT									✓
PROPCTL	✓	✓					✓	✓	
PUTAUT			✓	✓		✓		✓	
QMNAME					✓				
QSGDISP	✓	✓	✓	✓	✓	✓	✓	✓	
RCVDATA	✓	✓	✓	✓	✓	✓	✓	✓	
RCVEXIT	✓	✓	✓	✓	✓	✓	✓	✓	

Table 71. DEFINE and ALTER CHANNEL parameters (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNTCONN	SVRCONN	CLUSSDR	CLUSRCVR	MQTT
REPLACE	✓	✓	✓	✓	✓	✓	✓	✓	
SCYDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SCYEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SENDDATA	✓	✓	✓	✓	✓	✓	✓	✓	
SENDEXIT	✓	✓	✓	✓	✓	✓	✓	✓	
SEQWRAP	✓	✓	✓	✓			✓	✓	
SHARECNV					✓	✓			
SHORTRTY	✓	✓					✓	✓	
SHORTTMR	✓	✓					✓	✓	
SSLCAUTH		✓	✓	✓		✓		✓	✓
SSLCIPH [1]	✓	✓	✓	✓	✓	✓	✓	✓	✓ [1]
SSLCIPH									✓
SSLKEYR									✓
SSLPEER	✓	✓	✓	✓	✓	✓	✓	✓	
STATCHL	✓	✓	✓	✓			✓	✓	
TPNAME	✓	✓		✓	✓	✓	✓	✓	
TRPTYPE	✓	✓	✓	✓	✓	✓	✓	✓	✓
USECLTID									✓
USEDLQ	✓	✓	✓	✓			✓	✓	
USERID	✓	✓		✓	✓		✓		
XMITQ	✓	✓							

Note:

1. If SSLCIPH is used with MQTT channels, it means SSL Cipher Suite. For all other channel types, it means SSL CipherSpec. See SSLCIPH.

AFFINITY

Use the channel affinity attribute when client applications connect multiple times using the same

queue manager name. With the attribute, you can choose whether the client uses the same client channel definition for each connection. This attribute is intended to be used when multiple applicable channel definitions are available.

PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions. The list is based on the weightings, with any applicable `CLNTWGHT(0)` definitions first and in alphabetic order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful non-`CLNTWGHT(0)` definitions are moved to the end of the list. `CLNTWGHT(0)` definitions remain at the start of the list and are selected first for each connection. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT was modified since the list was created. Each client process with the same host name creates the same list.

NONE The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the weighting with any applicable `CLNTWGHT(0)` definitions selected first in alphabetic order. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT was modified since the list was created.

For example, suppose that we had the following definitions in the CCDT:

```
CHLNAME(A) QMNAME(QM1) CLNTWGHT(3)
CHLNAME(B) QMNAME(QM1) CLNTWGHT(4)
CHLNAME(C) QMNAME(QM1) CLNTWGHT(4)
```

The first connection in a process creates its own ordered list based on the weightings. So it might, for example, create the ordered list `CHLNAME(B), CHLNAME(A), CHLNAME(C)`.

For `AFFINITY(PREFFERED)`, each connection in the process attempts to connect using `CHLNAME(B)`. If a connection is unsuccessful the definition is moved to the end of the list which now becomes `CHLNAME(A), CHLNAME(C), CHLNAME(B)`. Each connection in the process then attempts to connect using `CHLNAME(A)`.

For `AFFINITY(NONE)`, each connection in the process attempts to connect using one of the three definitions selected at random based on the weightings.

If sharing conversations is enabled with a non-zero channel weighting and `AFFINITY(NONE)`, multiple connections do not have to share an existing channel instance. They can connect to the same queue manager name using different applicable definitions rather than sharing an existing channel instance.

BACKLOG(*integer*)
The number of outstanding connection requests that the telemetry channel can support at any one time. When the backlog limit is reached, any further clients trying to connect are refused connection until the current backlog is processed.

The value is in the range hyphen 0 - 999999999.

The default value is 4096.

BATCHHB(*integer*)
Specifies whether batch heartbeats are to be used. The value is the length of the heartbeat in milliseconds.

Batch heartbeats allow a sending channel to verify that the receiving channel is still active just before committing a batch of messages. If the receiving channel is not active, the batch can be backed out rather than becoming in-doubt, as would otherwise be the case. By backing out the batch, the messages remain available for processing so they could, for example, be redirected to another channel.

If the sending channel received a communication from the receiving channel within the batch heartbeat interval, the receiving channel is assumed to be still active. If not, a 'heartbeat' is sent to the receiving channel to check.

The value must be in the range 0 - 999999. A value of zero indicates that batch heart beats are not used.

This parameter is valid for channels with a channel type (CHLTYPE) of only SDR, SVR, CLUSSDR, and CLUSRCVR.

BATCHINT(*integer***)**

The minimum amount of time, in milliseconds, that a channel keeps a batch open.

The batch is terminated when one of the following conditions is met:

- BATCHSZ messages are sent.
- BATCHLIM kilobytes are sent.
- The transmission queue is empty and BATCHINT is exceeded.

The value must be in the range 0 - 999999999. Zero means that the batch is terminated as soon as the transmission queue becomes empty (or the BATCHSZ limit is reached).

This parameter is valid for channels with a channel type (CHLTYPE) of only SDR, SVR, CLUSSDR, and CLUSRCVR.

BATCHLIM(*integer***)**

The limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point. A sync point is taken after the message that caused the limit to be reached flows across the channel. A value of zero in this attribute means that no data limit is applied to batches over this channel.

The batch is terminated when one of the following conditions is met:

- BATCHSZ messages are sent.
- BATCHLIM kilobytes are sent.
- The transmission queue is empty and BATCHINT is exceeded.

This parameter is valid for channels with a channel type (CHLTYPE) of only SDR, SVR, CLUSSDR, and CLUSRCVR.

The value must be in the range 0 - 999999. The default value is 5000.

This parameter is supported on all platforms.

BATCHSZ (*integer*)

The maximum number of messages that can be sent through a channel before taking a sync point.

The maximum batch size used is the lowest of the following values:

- The BATCHSZ of the sending channel.
- The BATCHSZ of the receiving channel.
- On distributed platforms, the maximum number of uncommitted messages allowed at the sending queue manager (or one if this value is zero or less).
- On distributed platforms, the maximum number of uncommitted messages allowed at the receiving queue manager (or one if this value is zero or less).

While non-persistent messages sent over an NPMSPEED (FAST) channel are delivered to a queue immediately (without waiting for a complete batch), the messages still contribute to the batch size for a channel and, therefore, cause confirm flows to occur when BATCHSZ messages have flowed.

If the batch flows are causing a performance impact when moving only non-persistent messages, and NPMSPEED is set to FAST, you should consider setting the BATCHSZ to the maximum permissible value of 9999, and BATCHLIM to zero.

Additionally, setting BATCHINT to a high value, for example, 999999999 keeps each batch "open" for longer, even if there are no new messages waiting on the transmission queue.

The above settings minimize the frequency of confirm flows, but be aware that if any persistent messages are moved over a channel with these settings, there will be significant delays in the delivery of those persistent messages only.

The maximum number of uncommitted messages is specified by the MAXUMSGS parameter of the ALTER QMGR command. This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

The value must be in the range 1 - 9999.

(channel-name)
The name of the new channel definition.

This parameter is required on all types of channel. On CLUSSDR channels, it can take a different form to the other channel types. If your convention for naming CLUSSDR channels includes the name of the queue manager, you can define a CLUSSDR channel using the +QMNAME+ construction. After connection to the matching CLUSRCVR channel, WebSphere MQ substitutes the correct repository queue manager name in place of +QMNAME+ in the CLUSSDR channel definition. This facility applies to AIX, HP-UX, IBM i, Linux, Solaris, and Windows only; see Components of a cluster

The name must not be the same as any existing channel defined on this queue manager (unless REPLACE or ALTER is specified). On z/OS, CLNTCONN channel names can duplicate others.

The maximum length of the string is 20 characters, and the string must contain only valid characters; see Rules for naming IBM WebSphere MQ objects.

CHLTYPE
Channel type. This parameter is required. It must follow immediately after the *(channel-name)* parameter on all platforms except z/OS.

SDR Sender channel

SVR Server channel

RCVR Receiver channel

RQSTR Requester channel

CLNTCONN
Client-connection channel

SVRCONN
Server-connection channel

CLUSSDR
CLUSSDR channel.

CLUSRCVR
Cluster-receiver channel.

MQTT Telemetry channel

When a channel is defined using the **DEFINE** command, it is defined in a stopped state. However, for telemetry channels, the **DEFINE** command defines and attempts to start the channel, and the command might return an error from the start operation. While this error might look like a failure, the channel might still exist because the **DEFINE** command worked, but the start failed. An example of this behavior might be the definition of

multiple channels on the default port: the second definition fails with a `port in use` reason code, but the channel is successfully created.

Note: If you are using the `REPLACE` option, you cannot change the channel type.

CLNTWGHT

Set the client channel weighting attribute to select a client channel definition at random based on its weighting when more than one suitable definition is available. Specify a value in the range 0 - 99.

The special value 0 indicates that no random load balancing is performed and applicable definitions are selected in alphabetic order. To enable random load balancing the value can be in the range 1 - 99, where 1 is the lowest weighting and 99 is the highest.

If a client application issues MQCONN with a queue manager name of **name* a client channel definition can be selected at random. The chosen definition is randomly selected based on the weighting. Any applicable `CLNTWGHT(0)` definitions selected are selected first in alphabetic order. Randomness in the selection of client connection definitions is not guaranteed.

For example, suppose that we had the following two definitions in the CCDT:

```
CHLNAME(TO.QM1) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address1) CLNTWGHT(2)
CHLNAME(TO.QM2) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address2) CLNTWGHT(4)
```

A client MQCONN with queue manager name `*GRP1` would choose one of the two definitions based on the weighting of the channel definition. (A random integer 1 - 6 would be generated. If the integer was in the range 1 through 2, `address1` would be used otherwise `address2` would be used). If this connection was unsuccessful the client would then use the other definition.

The CCDT might contain applicable definitions with both zero and non-zero weighting. In this situation, the definitions with zero weighting are chosen first and in alphabetic order. If these connections are unsuccessful the definitions with non-zero weighting are chosen based on their weighting.

For example, suppose that we had the following four definitions in the CCDT:

```
CHLNAME(TO.QM1) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address1) CLNTWGHT(1)
CHLNAME(TO.QM2) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address2) CLNTWGHT(2)
CHLNAME(TO.QM3) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address3) CLNTWGHT(0)
CHLNAME(TO.QM4) CHLTYPE(CLNTCONN) QMNAME(GRP1) CONNAME(address4) CLNTWGHT(0)
```

A client MQCONN with queue manager name `*GRP1` would first choose definition `TO.QM3`. If this connection was unsuccessful the client would then choose definition `TO.QM4`. If this connection was also unsuccessful the client would then randomly choose one of the remaining two definitions based on their weighting.

CLNTWGHT is supported for all transport protocols.

CLUSNL(*nlname***)**

The name of the namelist that specifies a list of clusters to which the channel belongs.

This parameter is valid only for channels with a channel type (CHLTYPE) of `CLUSSDR` and `CLUSRCVR` channels. Only one of the resultant values of CLUSTER or CLUSNL can be nonblank, the other must be blank.

CLUSTER(*clustername***)**

The name of the cluster to which the channel belongs. The maximum length is 48 characters conforming to the rules for naming WebSphere MQ objects.

This parameter is valid only for channels with a channel type (CHLTYPE) of `CLUSSDR` and `CLUSRCVR` channels. Only one of the resultant values of CLUSTER or CLUSNL can be nonblank, the other must be blank.

CLWLPRTY(*integer*)
Specifies the priority of the channel for the purposes of cluster workload distribution. The value must be in the range 0 - 9 where 0 is the lowest priority and 9 is the highest.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLUSSDR and CLUSRCVR channels.

For more information about this attribute, see CLWLPRTY channel attribute.

CLWLRANK(*integer*)
Specifies the rank of the channel for the purposes of cluster workload distribution. The value must be in the range 0 - 9 where 0 is the lowest rank and 9 is the highest.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLUSSDR and CLUSRCVR channels.

For more information about this attribute, see CLWLRANK channel attribute.

CLWLWGHT(*integer*)
Specifies the weighting to be applied to a channel so that the proportion of messages sent down the channel can be controlled by workload management. The value must be in the range 1 - 99 where 1 is the lowest rank and 99 is the highest.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLUSSDR and CLUSRCVR channels.

For more information about this attribute, see CLWLWGHT channel attribute.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must either be left blank, or if QSGDISP is set to GROUP, the local queue manager name.

' ' The command is executed on the queue manager on which it was entered.

QmgrName
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which the command was entered. To do so, you must be using a shared queue environment, and the command server must be enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

COMPHDR
The list of header data compression techniques supported by the channel.

For SDR, SVR, CLUSSDR, CLUSRCVR, and CLNTCONN channels, the values are specified in order of preference. The first compression technique in the list that is supported by the remote end of the channel is used.
The mutually supported compression techniques of the channel are passed to the message exit of the sending channel. The message exit can alter the compression technique on a per message basis. Compression alters the data passed to send and receive exits.

NONE No header data compression is performed.

SYSTEM Header data compression is performed.

COMPMSG
The list of message data compression techniques supported by the channel.

For SDR, SVR, CLUSSDR, CLUSRCVR, and CLNTCONN channels, the values are specified in order of preference. The first compression technique in the list that is supported by the remote end of the channel is used.
The mutually supported compression techniques of the channel are passed to the message exit of the sending channel. The message exit can alter the compression technique on a per message basis. Compression alters the data passed to send and receive exits.

NONE No message data compression is performed.

RLE Message data compression is performed using run-length encoding.

ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

ANY Any compression technique supported by the queue manager can be used. This value is only valid for RCVR, RQSTR, and SVRCONN channels.

CONNAME(*string***)**
Connection name.

For CLUSRCVR channels, CONNAME relates to the local queue manager, and for other channels it relates to the target queue manager.

The maximum length of the string is 48 characters on z/OS, and 264 characters on other platforms.

A workaround to the 48 character limit might be one of the following suggestions:

- Set up your DNS servers so that you use, for example, host name of myserver instead of myserver.location.company.com, ensuring you can use the short host name.
- Use IP addresses.

Specify CONNAME as a comma-separated list of names of machines for the stated TRPTYPE. Typically only one machine name is required. You can provide multiple machine names to configure multiple connections with the same properties. The connections are usually tried in the order they are specified in the connection list until a connection is successfully established. The order is modified for clients if the CLNTWGHT attribute is provided. If no connection is successful, the channel attempts the connection again, as determined by the attributes of the channel. With client channels, a connection-list provides an alternative to using queue manager groups to configure multiple connections. With message channels, a connection list is used to configure connections to the alternative addresses of a multi-instance queue manager.

CONNAME is required for channels with a channel type (CHLTYPE) of SDR, RQSTR, CLNTCONN, and CLUSSDR. It is optional for SVR channels, and for CLUSRCVR channels of TRPTYPE(TCP), and is not valid for RCVR or SVRCONN channels.

Providing multiple connection names in a list was first supported in IBM WebSphere MQ Version 7.0.1. It changes the syntax of the CONNAME parameter. Earlier clients and queue managers connect using the first connection name in the list, and do not read the rest of the connection names in the list. In order for the earlier clients and queue managers to parse the new syntax, you must specify a port number on the first connection name in the list. Specifying a port number avoids problems when connecting to the channel from a client or queue manager that is running at a level earlier than IBM WebSphere MQ Version 7.0.1.

On AIX, HP-UX, IBM i, Linux, Solaris, and Windows platforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, WebSphere MQ generates a connection name for you, assuming the default port and using the

current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:

```
(1415)
```

The generated CONNAME is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.

Tip: If you are using any of the special characters in your connection name (for example, parentheses) you must enclose the string in single quotation marks.

The value you specify depends on the transport type (TRPTYPE) to be used:

LU62

- On z/OS, there are two forms in which to specify the value:

  Logical unit name
  The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. Logical unit name can be specified in one of three forms:

Form	Example
luname	IGY12355
luname/TPname	IGY12345/APING
luname/TPname/modename	IGY12345/APINGD/#INTER

  For the first form, the TP name and mode name must be specified for the TPNAME and MODENAME parameters; otherwise these parameters must be blank.

  Note: For CLNTCONN channels, only the first form is allowed.

  Symbolic name
  The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The TPNAME and MODENAME parameters must be blank.

  Note: For CLUSRCVR channels, the side information is on the other queue managers in the cluster. Alternatively, it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

  The specified or implied LU name can be that of a VTAM generic resources group.

- On AIX, HP-UX, IBM i, Linux, Solaris, and Windows, CONNAME is the name of the CPI-C communications side object. Alternatively, if the TPNAME is not blank, CONNAME is the fully qualified name of the partner logical unit.

NetBIOS
A unique NetBIOS name (limited to 16 characters).

SPX The 4-byte network address, the 6-byte node address, and the 2-byte socket number. These values must be entered in hexadecimal, with a period separating the network and node addresses. The socket number must be enclosed in brackets, for example:

```
CONNAME('0a0b0c0d.804abcde23a1(5e86)')
```

TCP Either the host name, or the network address of the remote machine (or the local machine for CLUSRCVR channels). This address can be followed by an optional port number, enclosed in parentheses.

If the CONNAME is a host name, the host name is resolved to an IP address.

The IP stack used for communication depends on the value specified for CONNAME and the value specified for LOCLADDR. See LOCLADDR for information about how this value is resolved.

On z/OS, the connection name can include the IP_name of an z/OS dynamic DNS group or a Network Dispatcher input port. Do not include the IP_name or input port for channels with a channel type (CHLTYPE) of CLUSSDR.

On AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS, you do not always need to specify the network address of your queue manager. If you define a channel with a channel type (CHLTYPE) of CLUSRCVR that is using TCP/IP, WebSphere MQ generates a CONNAME for you. It assumes the default port and uses the current IPv4 address of the system. If the system does not have an IPv4 address, the current IPv6 address of the system is used.

Note: If you are using clustering between IPv6-only and IPv4-only queue managers, do not specify an IPv6 network address as the CONNAME for CLUSRCVR channels. A queue manager that is capable only of IPv4 communication is unable to start a CLUSSDR channel definition that specifies the CONNAME in IPv6 hexadecimal form. Consider, instead, using host names in a heterogeneous IP environment.

CONVERT
Specifies whether the sending message channel agent attempts conversion of the application message data, if the receiving message channel agent cannot perform this conversion.

NO No conversion by sender

YES Conversion by sender

On z/OS, N and Y are accepted as synonyms of NO and YES.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

DEFCDISP
Specifies the default channel disposition of the channel.

PRIVATE
The intended disposition of the channel is as a private channel.

FIXSHARED
The intended disposition of the channel is as a shared channel associated with a specific queue manager.

SHARED The intended disposition of the channel is as a shared channel.

This parameter does not apply to channels with a channel type (CHLTYPE) of CLNTCONN, CLUSSDR, or CLUSRCVR.

DEFRECON
Specifies whether a client connection automatically reconnects a client application if its connection breaks.

NO Unless overridden by MQCONNX, the client is not reconnected automatically.

YES Unless overridden by MQCONNX, the client reconnects automatically.

QMGR Unless overridden by MQCONNX, the client reconnects automatically, but only to the same queue manager. The QMGR option has the same effect as MQCNO_RECONNECT_Q_MGR.

DISABLED
Reconnection is disabled, even if requested by the client program using the MQCONNX MQI call.

Table 72. Automatic reconnection depends on the values set in the application and in the channel definition

DEFRECON	Reconnection options set in the application			
	MQCNO_RECONNECT	MQCNO_RECONNECT_Q_MGR	MQCNO_RECONNECT_AS_DEF	MQCNO_RECONNECT_DISABLED
NO	YES	QMGR	NO	NO
YES	YES	QMGR	YES	NO
QMGR	YES	QMGR	QMGR	NO
DISABLED	NO	NO	NO	NO

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the channel when an operator issues the **DISPLAY CHANNEL** command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If the information is sent to another queue manager they might be translated incorrectly. The characters must be in the coded character set identifier (CCSID) of the local queue manager.

DISCINT(*integer***)**
The minimum time in seconds for which the channel waits for a message to arrive on the transmission queue. The waiting period starts after a batch ends. After the end of the waiting period, if there are no more messages, the channel is ended. A value of zero causes the message channel agent to wait indefinitely.

The value must be in the range 0 - 999 999.

This parameter is valid only for channels with a channel type (CHLTYPE) of SVRCONN, SDR, SVR, CLUSSDR, CLUSRCVR.

For SVRCONN channels using the TCP protocol, DISCINT has a different interpretation. It is the minimum time in seconds for which the SVRCONN instance remains active without any communication from its partner client. A value of zero disables this disconnect processing. The SVRCONN inactivity interval applies only between WebSphere MQ API calls from a client, so no client is disconnected during an extended MQGET with wait call. This attribute is ignored for SVRCONN channels using protocols other than TCP.

HBINT(*integer***)**

HBINT specifies the approximate time between heartbeat flows sent by a message channel agent (MCA). The flows are sent when there are no messages on the transmission queue.

Heartbeat flows unblock the receiving MCA, which is waiting for messages to arrive or for the disconnect interval to expire. When the receiving MCA is unblocked, it can disconnect the channel without waiting for the disconnect interval to expire. Heartbeat flows also free any storage buffers that are allocated for large messages. They also close any queues that are left open at the receiving end of the channel.

The value is in seconds and must be in the range 0 - 999999. A value of zero means that no heartbeat flows are to be sent. The default value is 300. To be most useful, the value needs to be less than the disconnect interval value.

For SVRCONN and CLNTCONN channels, heartbeats can flow from both the server side as well as the client side independently. If no data is transferred across the channel during the heartbeat interval, the CLNTCONN MQI agent sends a heartbeat flow. The SVRCONN MQI agent responds to it with another heartbeat flow. The flows happen irrespective of the state of the channel. For

example, irrespective of whether it is inactive while making an API call, or is inactive waiting for client user input. The SVRCONN MQI agent is also capable of initiating a heartbeat to the client, again irrespective of the state of the channel. The SVRCONN and CLNTCONN MQI agents are prevented from heart beating to each other at the same time. The server heartbeat is flowed if no data is transferred across the channel for the heartbeat interval plus 5 seconds.

Before IBM WebSphere MQ Version 7.0, for server-connection and client-connection channels working in the channel mode, heartbeats flow only when a server MCA is waiting for an MQGET command with the WAIT option specified, which it has issued on behalf of a client application.

For more information, see Heartbeat interval (HBINT).

JAASCFG(*string***)**
The name of a stanza in the JAAS configuration file.

KAINT(*integer***)**
The value passed to the communications stack for keepalive timing for this channel.

For this attribute to be effective, TCP/IP keepalive must be enabled both in the queue manager and in TCP/IP. On z/OS, enable TCP/IP keepalive in the queue manager by issuing the ALTER QMGR TCPKEEP(YES) command. If the TCPKEEP queue manager parameter is NO, the value is ignored, and the keepalive facility is not used. On other platforms, TCP/IP keepalive is enabled when the KEEPALIVE=YES parameter is specified in the TCP stanza. Modify the TCP stanza in the distributed queuing configuration file, qm.ini, or through the WebSphere MQ Explorer.

Keepalive must also be switched on within TCP/IP itself. Refer to your TCP/IP documentation for information about configuring keepalive. On AIX, use the **no** command. On HP-UX, use the **ndd** command. On Windows, edit the registry. On z/OS, update your TCP/IP PROFILE data set and add or change the INTERVAL parameter in the TCPCONFIG section.

Although this parameter is available on all platforms, its setting is implemented only on z/OS. On platforms other than z/OS, you can access and modify the parameter, but it is only stored and forwarded. It is not implemented, but it is still useful, for instance in a clustered environment. For example, a value set in a CLUSRCVR channel definition on Solaris flows to z/OS queue managers that are in, or join, the cluster.

On platforms other than z/OS, if you need the functionality provided by the KAINT parameter, use the Heartbeat Interval (HBINT) parameter, as described in HBINT.

(*integer***)**
The KeepAlive interval to be used, in seconds, in the range 1 through 99999.

0 The value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

AUTO The KeepAlive interval is calculated based upon the negotiated heartbeat value as follows:

- If the negotiated HBINT is greater than zero, keepalive interval is set to that value plus 60 seconds.
- If the negotiated HBINT is zero, the keepalive value used is that specified by the INTERVAL statement in the TCP/IP PROFILE configuration data set.

If AUTO is specified for KAINT, and it is a server-connection channel, the TCP INTERVAL value is used instead for the keepalive interval.

In this case, KAINT is zero in DISPLAY CHSTATUS; it would be non-zero if an integer had been coded instead of AUTO.

This parameter is valid for all channel types. It is ignored for channels with a TRPTYPE other than TCP or SPX.

LIKE(*channel-name***)**
The name of a channel. The parameters of this channel are used to model this definition.

If you do not set LIKE, and do not set a parameter field related to the command, its value is taken from one of the default channels. The default values depend upon the channel type:

SYSTEM.DEF.SENDER
Sender channel

SYSTEM.DEF.SERVER
Server channel

SYSTEM.DEF.RECEIVER
Receiver channel

SYSTEM.DEF.REQUESTER
Requester channel

SYSTEM.DEF.SVRCONN
Server-connection channel

SYSTEM.DEF.CLNTCONN
Client-connection channel

SYSTEM.DEF.CLUSSDR
CLUSSDR channel

SYSTEM.DEF.CLUSRCVR
Cluster-receiver channel

SYSTEM.DEF.MQTT
Telemetry channel

This parameter is equivalent to defining the following object:
`LIKE(SYSTEM.DEF.SENDER)`

for a SDR channel, and similarly for other channel types.

These default channel definitions can be altered by the installation to the default values required.

On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object and channel type you are defining.

Note:

1. QSGDISP(GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified. However, the group object defined is used as a LIKE object.

LOCLADDR(*string*)
LOCLADDR is the local communications address for the channel. Use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. LOCLADDR might be useful in recovery scenarios where a channel is restarted on a different TCP/IP stack. LOCLADDR is also useful to force a channel to use an IPv4 or IPv6 stack on a dual-stack system. You can also use LOCLADDR to force a channel to use a dual-mode stack on a single-stack system.

This parameter is valid only for channels with a transport type (TRPTYPE) of TCP. If TRPTYPE is not TCP, the data is ignored and no error message is issued.

The value is the optional IP address, and optional port or port range used for outbound TCP/IP communications. The format for this information is as follows:

Table 73 shows how the LOCLADDR parameter can be used:
`LOCLADDR([ip-addr][(low-port[,high-port])]`*`[,[ip-addr][(low-port[,high-port])]]`*`)`

The maximum length of LOCLADDR, including multiple addresses, is `MQ_LOCAL_ADDRESS_LENGTH`.

If you omit LOCLADDR, a local address is automatically allocated.

Note, that you can set LOCLADDR for a C client using the Client Channel Definition Table (CCDT).

All the parameters are optional. Omitting the `ip-addr` part of the address is useful to enable the configuration of a fixed port number for an IP firewall. Omitting the port number is useful to select a particular network adapter without having the identify a unique local port number. The TCP/IP stack generates a unique port number.

Specify *`[,[ip-addr][(low-port[,high-port])]]`* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *`[,[ip-addr][(low-port[,high-port])]]`* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

`ip-addr`
`ip-addr` is specified in one of three forms:

IPv4 dotted decimal
For example `192.0.2.1`

IPv6 hexadecimal notation
For example `2001:DB8:0:0:0:0:0:0`

Alphanumeric host name form
For example `WWW.EXAMPLE.COM`

`low-port and high-port`
`low-port` and `high-port` are port numbers enclosed in parentheses.

Table 73. Examples of how the LOCLADDR parameter can be used

LOCLADDR	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98, 9.20.4.99	Channel binds to either IP address. The address might be two network adapters on one server, or a different network adapter on two different servers in a multi-instance configuration.
9.20.4.98(1000)	Channel binds to this address and port 1000 locally
9.20.4.98(1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to port in range 1000 - 2000 locally

This parameter is valid only for channels with a channel type (CHLTYPE) of `SDR`, `SVR`, `RQSTR`, `CLNTCONN`, `CLUSSDR`, `CLUSRCVR`, or `MQTT`.

On `CLUSSDR` channels, the IP address and port to which the outbound channel binds, is a combination of fields. It is a concatenation of the IP address, as defined in the LOCLADDR parameter, and the port range from the cluster cache. If there is no port range in the cache, the port range defined in the LOCLADDR parameter is used. This port range does not apply to z/OS.

Even though this parameter is similar in form to CONNAME, it must not be confused with it. The LOCLADDR parameter specifies the characteristics of the local communications, whereas the CONNAME parameter specifies how to reach a remote queue manager.

When a channel is started, the values specified for CONNAME and LOCLADDR determine the IP stack to be used for communication; see Table 3 and Local Address (LOCLADDR) .

If the TCP/IP stack for the local address is not installed or configured, the channel does not start and an exception message is generated. The message indicates that the connect() request specifies an interface address that is not known on the default IP stack. To direct the connect() request to the alternative stack, specify the **LOCLADDR** parameter in the channel definition as either an interface on the alternative stack, or a DNS host name. The same specification also works for listeners that might not use the default stack. To find the value to code for **LOCLADDR**, run the **NETSTAT HOME** command on the IP stacks that you want to use as alternatives.

For channels with a channel type (CHLTYPE) of MQTT the usage of this parameter is slightly different. Specifically, a telemetry channel (MQTT) **LOCLADDR** parameter expects only an IPv4 or IPv6 IP address, or a valid host name as a string. This string must not contain a port number or port range. If an IP address is entered, only the address format is validated. The IP address itself is not validated.

Table 74. How the IP stack to be used for communication is determined

Protocols supported	CONNAME	LOCLADDR	Action of channel
IPv4 only	IPv4 address[1]		Channel binds to IPv4 stack
	IPv6 address[2]		Channel fails to resolve CONNAME
	IPv4 and 6 host name[3]		Channel binds to IPv4 stack
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	Any address[4]	IPv6 address	Channel fails to resolve LOCLADDR
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv4 stack
IPv4 and IPv6	IPv4 address		Channel binds to IPv4 stack
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to stack determined by IPADDRV
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	IPv4 address	IPv6 address	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv6 address	Channel binds IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to stack determined by IPADDRV

Table 74. How the IP stack to be used for communication is determined (continued)

Protocols supported	CONNAME	LOCLADDR	Action of channel
IPv6 only	IPv4 address		Channel maps CONNAME to IPv6[5]
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to IPv6 stack
	Any address	IPv4 address	Channel fails to resolve LOCLADDR
	IPv4 address	IPv6 address	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv6 address	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds to IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv6 stack

Notes:

1. IPv4 address. An IPv4 host name that resolves only to an IPv4 network address or a specific dotted notation IPv4 address, for example `1.2.3.4`. This note applies to all occurrences of 'IPv4 address' in this table.
2. IPv6 address. An IPv6 host name that resolves only to an IPv6 network address or a specific hexadecimal notation IPv6 address, for example `4321:54bc`. This note applies to all occurrences of 'IPv6 address' in this table.
3. IPv4 and 6 host name. A host name that resolves to both IPv4 and IPv6 network addresses. This note applies to all occurrences of 'IPv4 and 6 host name' in this table.
4. Any address. IPv4 address, IPv6 address, or IPv4 and 6 host name. This note applies to all occurrences of 'Any address' in this table.
5. Maps IPv4 CONNAME to IPv4 mapped IPv6 address. IPv6 stack implementations that do not support IPv4 mapped IPv6 addressing fail to resolve the CONNAME. Mapped addresses might require protocol translators in order to be used. The use of mapped addresses is not recommended.

LONGRTY(*integer*)

The LONGRTY parameter specifies the maximum number of further attempts that are made by a `SDR`, `SVR`, or `CLUSSDR` channel to connect to a remote queue manager. The interval between attempts is specified by LONGTMR. The LONGRTY parameter takes effect if the count specified by SHORTRTY is exhausted.

If this count is exhausted without success, an error is logged to the operator, and the channel stops. In this circumstance, the channel must be restarted with a command. It is not started automatically by the channel initiator.

The LONGRTY value must be in the range 0 - 9999999.

This parameter is valid only for channels with a channel type (CHLTYPE) of `SDR`, `SVR`, `CLUSSDR`, or `CLUSRCVR`.

A channel attempts to reconnect if it fails to connect initially, whether it is started automatically by the channel initiator or by an explicit command. It also tries to connect again if the connection fails after the channel successfully connecting. If the cause of the failure is such that more attempts are unlikely to be successful, they are not attempted.

LONGTMR(*integer*)

For LONGRTY, LONGTMR is the maximum number of seconds to wait before reattempting connection to the remote queue manager.

The time is approximate; zero means that another connection attempt is made as soon as possible.

The interval between attempting to reconnect might be extended if the channel has to wait to become active.

The LONGTMR value must be in the range 0 - 9999999.

Note: For implementation reasons, the maximum LONGTMR value is 999,999; values exceeding this maximum are treated as 999,999. Similarly, the minimum interval between attempting to reconnect is 2 seconds. Values less than this minimum are treated as 2 seconds.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

MAXINST(*integer*)
The maximum number of simultaneous instances of an individual SVRCONN channel that can be started.

The value must be in the range 0 - 999999999.

A value of zero prevents all client access on this channel.

New instances cannot start if the number of running instances equals or exceeds the value of this parameter. If MAXINST is changed to less than the number of instances of the SVRCONN channel that are currently running, the number of running instances is not affected.

On z/OS, without the client attachment feature installed, a maximum of five instances are allowed on the SYSTEM.ADMIN.SVRCONN channel. If MAXINST is set to a larger number than five, it is interpreted as zero without the CAF installed.

This parameter is valid only for channels with a channel type (CHLTYPE) of SVRCONN.

MAXINSTC(*integer*)
The maximum number of simultaneous individual SVRCONN channels that can be started from a single client. In this context, connections that originate from the same remote network address are regarded as coming from the same client.

The value must be in the range 0 - 999999999.

A value of zero prevents all client access on this channel.

If you reduce the value of MAXINSTC to less than the number of instances of the SVRCONN channel that is currently running from an individual client, the running instances are not affected. New SVRCONN instances from that client cannot start until the client is running fewer instances than the value of MAXINSTC.

On z/OS, without the client attachment feature installed, only a maximum of five instances are allowed on the channel named SYSTEM.ADMIN.SVRCONN.

This parameter is valid only for channels with a channel type (CHLTYPE) of SVRCONN.

MAXMSGL(*integer*)
Specifies the maximum message length that can be transmitted on the channel. This parameter is compared with the value for the partner and the actual maximum used is the lower of the two values. The value is ineffective if the MQCB function is being executed and the channel type (CHLTYPE) is SVRCONN.

The value zero means the maximum message length for the queue manager; see ALTER QMGR MAXMSGL.

On AIX, HP-UX, IBM i, Linux, Solaris, and Windows, specify a value in the range zero to the maximum message length for the queue manager.

On z/OS, specify a value in the range 0 - 104857600 bytes (100 MB).

Note that by adding the digital signature and key to the message, WebSphere MQ Advanced Message Security increases the length of the message.

MCANAME(*string*)
Message channel agent name.

This parameter is reserved, and if specified must be set to blanks (maximum length 20 characters).

MCATYPE

Specifies whether the message-channel-agent program on an outbound message channel runs as a thread or a process.

PROCESS
The message channel agent runs as a separate process.

THREAD The message channel agent runs as a separate thread

In situations where a threaded listener is required to service many incoming requests, resources can become strained. In this case, use multiple listener processes and target incoming requests at specific listeners though the port number specified on the listener.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLUSSDR, or CLUSRCVR. It is supported only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

On z/OS, it is supported only for channels with a channel type of CLUSRCVR. When specified in a CLUSRCVR definition, MCATYPE is used by a remote machine to determine the corresponding CLUSSDR definition.

MCAUSER(*string*)

Message channel agent user identifier.

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL). For more details, see Channel authentication records

This parameter interacts with PUTAUT; see PUTAUT.

If MCAUSER is nonblank, a user identifier is used by the message channel agent for authorization to access WebSphere MQ resources. If PUTAUT is DEF, authorization includes authorization to put the message to the destination queue for RCVR or RQSTR channels.

If it is blank, the message channel agent uses its default user identifier.

The default user identifier is derived from the user ID that started the receiving channel. The possible values are:

z/OS, The user ID assigned to the channel-initiator started task by the z/OS started-procedures table.

TCP/IP, other than z/OS
The user ID from the inetd.conf entry, or the user that started the listener.

SNA, other than z/OS
The user ID from the SNA server entry. In the absence of the user ID from the SNA server entry, the user from the incoming attach request, or the user that started the listener.

NetBIOS or SPX
The user ID that started the listener.

The maximum length of the string is 64 characters on Windows and 12 characters on other platforms. On Windows, you can optionally qualify a user identifier with the domain name in the format user@domain.

This parameter is not valid for channels with a channel type (CHLTYPE) of SDR, SVR, CLNTCONN, CLUSSDR.

MODENAME(*string*)
LU 6.2 mode name (maximum length 8 characters).

This parameter is valid only for channels with a transport type (TRPTYPE) of LU62. If TRPTYPE is not LU62, the data is ignored and no error message is issued.

If specified, this parameter must be set to the SNA mode name unless the CONNAME contains a side-object name. If CONNAME is a a side-object name it must be set to blanks. The actual name is then taken from the CPI-C Communications Side Object, or APPC side information data set.

This parameter is not valid for channels with a channel type (CHLTYPE) of RCVR or SVRCONN.

MONCHL
Controls the collection of online monitoring data for channels:

QMGR Collect monitoring data according to the setting of the queue manager parameter MONCHL.

OFF Monitoring data collection is turned off for this channel.

LOW If the value of the queue manager MONCHL parameter is not NONE, online monitoring data is turned on. Data us collected at a low rate for this channel.

MEDIUM If the value of the queue manager MONCHL parameter is not NONE, online monitoring data is turned on. Data us collected at a medium rate for this channel.

HIGH If the value of the queue manager MONCHL parameter is not NONE, online monitoring data is turned on. Data us collected at a high rate for this channel.

Changes to this parameter take effect only on channels started after the change occurs.

For cluster channels, the value of this parameter is not replicated in the repository and, therefore, not used in the auto-definition of CLUSSDR channels. For auto-defined CLUSSDR channels, the value of this parameter is taken from the queue manager attribute MONACLS. This value might then be overridden in the channel auto-definition exit.

MRDATA(*string*)
Channel message-retry exit user data. The maximum length is 32 characters.

This parameter is passed to the channel message-retry exit when it is called.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, or CLUSRCVR.

MREXIT(*string*)
Channel message-retry exit name.

The format and maximum length of the name is the same as for MSGEXIT, however you can specify only one message-retry exit.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, or CLUSRCVR.

MRRTY(*integer*)
The number of times the channel tries again before it decides it cannot deliver the message.

This parameter controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRRTY is passed to the exit to use. The number of attempts to redeliver the message is controlled by the exit, and not by this parameter.

The value must be in the range 0 - 999999999. A value of zero means that no attempts to redeliver the message are tried.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, or CLUSRCVR.

MRTMR(*integer*)
The minimum interval of time that must pass before the channel can try the MQPUT operation again. The time interval is in milliseconds.

This parameter controls the action of the MCA only if the message-retry exit name is blank. If the exit name is not blank, the value of MRTMR is passed to the exit to use. The number of attempts to redeliver the message is controlled by the exit, and not by this parameter.

The value must be in the range 0 - 999999999. A value of zero means that if the value of MRRTY is greater than zero, the channel reattempts delivery as soon as possible.

This parameter is valid only for channels with a channel type (CHLTYPE) of `RCVR`, `RQSTR`, or `CLUSRCVR`.

MSGDATA(*string*)
User data for the channel message exit. The maximum length is 32 characters.

This data is passed to the channel message exit when it is called.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first message exit specified, the second string to the second exit, and so on.

On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first message exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of message exit data for each channel.

Note: This parameter is accepted but ignored for `SVRCONN` and `CLNTCONN` channels.

MSGEXIT(*string*)
Channel message exit name.

If MSGEXIT is nonblank the exit is called at the following times:

- Immediately after a `SDR` or `SVR` channel retrieves a message from the transmission queue.
- Immediately before a `RQSTR` channel puts a message on destination queue.
- When the channel is initialized or ended.

The exit is passed the entire application message and transmission queue header for modification.

MSGEXIT is accepted and ignored by `CLNTCONN` and `SVRCONN` channels. `CLNTCONN` or `SVRCONN` channels do not call message exits.

The format and maximum length of the exit name depends on the platform; see Table 75 on page 459.

If the MSGEXIT, MREXIT, SCYEXIT, SENDEXIT, and RCVEXIT parameters are all left blank, the channel user exit is not invoked. If any of these parameters is nonblank, the channel exit program is called. You can enter text string for these parameters. The maximum length of the string is 128 characters.

Table 75. Message exit format and length

Platform	Exit name format	Maximum length	Comment
AIX, HP-UX, Linux, and Solaris	*libraryname*(*functionname*)	128	You can specify the name of more than one exit program. Specify multiple strings separated by commas. However, the total number of characters specified must not exceed 999.
Windows	*dllname*(*functionname*)	128	1. You can specify the name of more than one exit program. Specify multiple strings separated by commas. However, the total number of characters specified must not exceed 999. 2. *dllname* is specified without the suffix (.DLL).
IBM i	*progname libname*	20	1. You can specify the names of up to 10 exit programs by specifying multiple strings separated by commas. 2. *program name* occupies the first 10 characters and *libname* the second 10 characters. If necessary, both fields are padded to the right with blanks.
z/OS	*loadModuleName*	8	1. You can specify the names of up to eight exit programs by specifying multiple strings separated by commas. 2. 128 characters are allowed for exit names for CLNTCONN channels, subject to a maximum total length including commas of 999.

- On systems, it is of the form:

NETPRTY(*integer***)**
The priority for the network connection. Distributed queuing chooses the path with the highest priority if there are multiple paths available. The value must be in the range 0 - 9; 0 is the lowest priority.

This parameter is valid only for CLUSRCVR channels.

NPMSPEED
The class of service for nonpersistent messages on this channel:

FAST Fast delivery for nonpersistent messages; messages might be lost if the channel is lost. Messages are retrieved using MQGMO_SYNCPOINT_IF_PERSISTENT and so are not included in the batch unit of work.

NORMAL Normal delivery for nonpersistent messages.

If the value of NPMSPEED differs between the sender and receiver, or either one does not support it, NORMAL is used.

This parameter is valid only for channels with a CHLTYPE of SDR, SVR, RCVR, RQSTR, CLUSSDR, or CLUSRCVR.

PASSWORD(*string***)**
Password used by the message channel agent when attempting to initiate a secure LU 6.2 session with a remote message channel agent. The maximum length is 12 characters.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLNTCONN, or CLUSSDR. On z/OS, it is supported only for channels with a channel type (CHLTYPE) of CLNTCONN.

Although the maximum length of the parameter is 12 characters, only the first 10 characters are used.

PORT(*integer*)
The port number for TCP/IP. This parameter is the port number on which the listener is to stop listening. It is valid only if the transmission protocol is TCP/IP.

PROPCTL
Property control attribute; see **PROPCTL** channel options.

PROPCTL specifies what happens to message properties when a message is sent to another queue manager; see

This parameter is applicable to SDR, SVR, CLUSSDR, and CLUSRCVR channels.

This parameter is optional.

Permitted values are:

COMPAT COMPAT allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

Message properties	Result
The message contains a property with a prefix of mcd., jms., usr. or mqext.	If the Support value is MQPD_SUPPORT_OPTIONAL, all optional message properties are placed in one or more MQRFH2 headers. This rule does not apply to properties in the message descriptor or extension, which remain in the same place. Optional message properties are moved into the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of mcd., jms., usr. or mqext.	All message properties, except properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The message contains a property where the Support field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL	The message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.
The message contains one or more properties where the Support field of the property descriptor is set to MQPD_SUPPORT_OPTIONAL. Other fields of the property descriptor are set to non-default values.	The properties with non-default values are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the content='properties' attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a Version 6.0 or prior queue manager.

NONE All properties of the message, except properties in the message descriptor or extension, are removed from the message. The properties are removed before the message is sent to the remote queue manager.

If the message contains a property where the Support field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL then the message is rejected with reason MQRC_UNSUPPORTED_PROPERTY. The error is reported in accordance with the report options set in the message header.

ALL All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

PUTAUT

PUTAUT specifies which user identifiers are used to establish authority for a channel. It specifies the user identifier to put messages to the destination queue using a message channel, or to run an MQI call using an MQI channel.

DEF The default user ID is used. On z/OS, DEF might involve using both the user ID received from the network and that derived from MCAUSER.

CTX The user ID from the *UserIdentifier* field of the message descriptor is used. On z/OS, CTX might involve also using the user ID received from the network or that derived from MCAUSER, or both.

ONLYMCA
The default user ID is used. Any user ID received from the network is not used. This value is supported only on z/OS.

ALTMCA The user ID from the *UserIdentifier* field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS.

On z/OS, the user IDs that are checked, and how many user IDs are checked, depends on the setting of the MQADMIN RACF class hlq.RESLEVEL profile. Depending on the level of access the user ID of the channel initiator has to hlq.RESLEVEL, zero, one, or two user IDs are checked.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, RQSTR, CLUSRCVR, or, on z/OS only, SVRCONN. CTX and ALTMCA are not valid for SVRCONN channels.

QMNAME(*string*)

Queue manager name.

For CLNTCONN channels, QMNAME is the name of a queue manager to which a IBM WebSphere MQ MQI client application can request connection. QMNAME is not necessarily the same as the name of the queue manager on which the channel is defined; see Queue manager groups in the CCDT.

For channels of other types, the QMNAME parameter is not valid.

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the LIKE object.

GROUP	The object definition resides in the shared repository but only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated. The command is sent to all active queue managers in the queue-sharing group to make or refresh local copies on page set zero:
	`DEFINE CHANNEL(channe-name)`
	`CHLTYPE(type) REPLACE QSGDISP(COPY)`
	
	The **DEFINE** command for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

RCVDATA(*string*)

Channel receive exit user data (maximum length 32 characters).

This parameter is passed to the channel receive exit when it is called.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first receive exit specified, the second string to the second exit, and so on.

On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first receive exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of receive exit data for each channel.

RCVEXIT(*string*)
Channel receive exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before the received network data is processed.

  The exit is given the complete transmission buffer as received. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one receive exit name for each channel.

The format and maximum length of the name is the same as for MSGEXIT.

REPLACE and NOREPLACE
Replace the existing definition with this one, or not. This parameter is optional. On z/OS it must have the same disposition. Any object with a different disposition is not changed.

REPLACE
The definition replaces any existing definition of the same name. If a definition does not exist, one is created. REPLACE does not alter the channel status.

NOREPLACE
The definition does not replace any existing definition of the same name.

SCYDATA(*string*)
Channel security exit user data (maximum length 32 characters).

This parameter is passed to the channel security exit when it is called.

SCYEXIT(*string*)
Channel security exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is able to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote queue manager are given to the exit.

- At initialization and termination of the channel.

The format and maximum length of the name is the same as for MSGEXIT but only one name is allowed.

SENDDATA(*string***)**
Channel send exit user data. The maximum length is 32 characters.

This parameter is passed to the channel send exit when it is called.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify data for more than one exit program by specifying multiple strings separated by commas. The total length of the field must not exceed 999 characters.

On IBM i, you can specify up to 10 strings, each of length 32 characters. The first string of data is passed to the first send exit specified, the second string to the second exit, and so on.

On z/OS, you can specify up to eight strings, each of length 32 characters. The first string of data is passed to the first send exit specified, the second string to the second exit, and so on.

On other platforms, you can specify only one string of send exit data for each channel.

SENDEXIT(*string***)**
Channel send exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before data is sent out on the network.

  The exit is given the complete transmission buffer before it is transmitted. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

On AIX, HP-UX, Linux, Solaris, and Windows, you can specify the name of more than one exit program by specifying multiple strings separated by commas. However, the total number of characters specified must not exceed 999.

On IBM i, you can specify the names of up to 10 exit programs by specifying multiple strings separated by commas.

On z/OS, you can specify the names of up to eight exit programs by specifying multiple strings separated by commas.

On other platforms, you can specify only one send exit name for each channel.

The format and maximum length of the name is the same as for MSGEXIT.

SEQWRAP(*integer***)**
When this value is reached, sequence numbers wrap to start again at 1.

This value is nonnegotiable and must match in both the local and remote channel definitions.

The value must be in the range 100 - 999999999.

This parameter is valid only for channels with a channel type (CHLTYPE) of `SDR`, `SVR`, `RCVR`, `RQSTR`, `CLUSSDR`, or `CLUSRCVR`.

SHARECNV(*integer***)**
Specifies the maximum number of conversations that can be sharing each TCP/IP channel instance. A SHARECNV value of:

1 Specifies no sharing of conversations over a TCP/IP channel instance. Client heart beating is available whether in an MQGET call or not. Read ahead and client asynchronous consumption are also available, and channel quiescing is more controllable.

0 Specifies no sharing of conversations over a TCP/IP channel instance. The channel instance runs in a mode that is compatible with WebSphere MQ earlier than version 7.0, regarding:

- Administrator stop-quiesce
- Heart beating
- Read ahead
- Client asynchronous consumption

The value must be in the range zero through 999999999.

This parameter is valid only for channels with a channel type (CHLTYPE) of CLNTCONN or SVRCONN. If the CLNTCONN SHARECNV value does not match the SVRCONN SHARECNV value, the lower of the two values is used. This parameter is ignored for channels with a transport type (TRPTYPE) other than TCP.

All the conversations on a socket are received by the same thread.

High SHARECNV limits have the advantage of reducing queue manager thread usage. If many conversations sharing a socket are all busy, there is a possibility of delays. The conversations contend with one another to use the receiving thread. In this situation, a lower SHARECNV value is better.

The number of shared conversations does not contribute to the MAXINST or MAXINSTC totals.

Note: You should restart the client for this change to take effect.

SHORTRTY(*integer*)
SHORTRTY specifies the maximum number of attempts that are made by a SDR, SVR, or CLUSSDR channel to connect to the remote queue manager, at intervals specified by SHORTTMR. After the number of attempts is exhausted, the channel tries to reconnect using to the schedule defined by LONGRTY.

The value must be in the range 0 - 999999999.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

A channel attempts to reconnect if it fails to connect initially, whether it is started automatically by the channel initiator or by an explicit command. It also tries to connect again if the connection fails after the channel successfully connecting. If the cause of the failure is such that more attempts are unlikely to be successful, they are not attempted.

SHORTTMR(*integer*)
For SHORTRTY, SHORTTMR is the maximum number of seconds to wait before reattempting connection to the remote queue manager.

The time is approximate.

The interval between attempting to reconnect might be extended if the channel has to wait to become active.

The value must be in the range 0 - 999999999.

Note: For implementation reasons, the maximum SHORTTMR value is 999,999; values exceeding this maximum are treated as 999,999. The minimum interval between attempting to connect is 10 seconds with SHORTTMR(0) and 2 seconds with SHORTTMR(1).

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, CLUSSDR, or CLUSRCVR.

SSLCAUTH
SSLCAUTH defines whether WebSphere MQ requires a certificate from the SSL client. The SSL client is the initiating end of the channel. SSLCAUTH is applied to the SSL server, to determine the behavior required of the client. The SSL server is the end of the channel that receives the initiation flow.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, SVRCONN, CLUSRCVR, SVR, RQSTR, or MQTT.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

REQUIRED
WebSphere MQ requires and validates a certificate from the SSL client.

OPTIONAL
The peer SSL client system might still send a certificate. If it does, the contents of this certificate are validated as normal.

SSLCIPH*(string)*
SSLCIPH specifies the CipherSpec that is used on the channel. The maximum length is 32 characters. This parameter is valid on all channel types which use transport type TRPTYPE(TCP). If the SSLCIPH parameter is blank, no attempt is made to use SSL on the channel.

Note: When SSLCIPH is used with a telemetry channel, it means "SSL Cipher Suite". See the SSLCIPH description in "DEFINE CHANNEL (MQTT)".

Specify the name of the CipherSpec you are using. The CipherSpecs that can be used with WebSphere MQ SSL support are shown in the following table. The SSLCIPH values must specify the same CipherSpec on both ends of the channel.

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
NULL_MD5 [a]	SSL 3.0	MD5	None	0	No	No	No
NULL_SHA [a]	SSL 3.0	SHA-1	None	0	No	No	No
RC4_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC4	40	No	No	No
RC4_MD5_US [a]	SSL 3.0	MD5	RC4	128	No	No	No
RC4_SHA_US [a]	SSL 3.0	SHA-1	RC4	128	No	No	No
RC2_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC2	40	No	No	No
DES_SHA_EXPORT [2] [a]	SSL 3.0	SHA-1	DES	56	No	No	No
RC4_56_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	RC4	56	No	No	No
DES_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	DES	56	No	No	No
TLS_RSA_WITH_AES_128_CBC_SHA [a]	TLS 1.0	SHA-1	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA [4] [a]	TLS 1.0	SHA-1	AES	256	Yes	No	No
TLS_RSA_WITH_DES_CBC_SHA [a]	TLS 1.0	SHA-1	DES	56	No[5]	No	No
FIPS_WITH_DES_CBC_SHA [b]	SSL 3.0	SHA-1	DES	56	No[6]	No	No
TLS_RSA_WITH_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	256	Yes	No	No
ECDHE_ECDSA_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No
ECDHE_RSA_RC4_128_SHA256 [b]	TLS 1.2	SHA_1	RC4	128	No	No	No

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
ECDHE_ECDSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_ECDSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_RSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_RSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_ECDSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	Yes	No
ECDHE_ECDSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	Yes
ECDHE_RSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
ECDHE_RSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_NULL_SHA256 [b]	TLS 1.2	SHA-256	None	0	No	No	No
ECDHE_RSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No
ECDHE_ECDSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No
TLS_RSA_WITH_NULL_NULL [b]	TLS 1.2	None	None	0	No	No	No
TLS_RSA_WITH_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No

Notes:

1. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS.
2. The maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
3. The handshake key size is 1024 bits.
4. This CipherSpec cannot be used to secure a connection from the WebSphere MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer.
5. This CipherSpec was FIPS 140-2 certified before 19 May 2007.
6. This CipherSpec was FIPS 140-2 certified before 19 May 2007. The name FIPS_WITH_DES_CBC_SHA is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. This CipherSpec is deprecated and its use is not recommended.
7. This CipherSpec can be used to transfer up to 32 GB of data before the connection is terminated with error AMQ9288. To avoid this error, either avoid using triple DES, or enable secret key reset when using this CipherSpec.

Platform support:

- a Available on all supported platforms.
- b Available only on UNIX, Linux, and Windows platforms.

When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the SSL handshake can depend on the size stored in the certificate and on the CipherSpec:

- On z/OS, Windows, UNIX and Linux systems, when a CipherSpec name includes _EXPORT, the maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
- On Windows, UNIX and Linux systems, when a CipherSpec name includes _EXPORT1024, the handshake key size is 1024 bits.
- Otherwise the handshake key size is the size stored in the certificate.

SSLKEYP(*string*)
The store for digital certificates and their associated private keys. If you do not specify a key file, SSL is not used.

SSLKEYR(*string*)
The password for the key repository. If no passphrase is entered, then unencrypted connections must be used.

SSLPEER*(string)*

Specifies the certificate filter used by the peer queue manager or client at the other end of the channel. The filter is used to compare with the distinguished name of the certificate. A "distinguished name" is the identifier of the SSL certificate. If the distinguished name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

Note: An alternative way of restricting connections into channels by matching against the SSL or TLS Subject distinguished name, is to use channel authentication records. With channel authentication records, different SSL or TLS subject distinguished name patterns can be applied to the same channel. Both SSLPEER and a channel authentication record can be applied to the same channel. If so, the inbound certificate must match both patterns in order to connect. For more information, see Channel authentication records.

SSLPEER is optional. If it is not specified, the distinguished name of the peer is not checked at channel startup. The distinguished name from the certificate is still written into the SSLPEER definition held in memory, and passed to the security exit. If SSLCIPH is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The SSLPEER value is specified in the standard form used to specify a distinguished name. For example:

```
SSLPEER('SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN="H1_C_FR1",O=IBM,C=GB')
```

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

Table 76. Attribute types supported by SSLPEER.

A two column table describing the attributes supported by the SSLPEER parameter

Attribute	Description
SERIALNUMBER	Certificate serial number
MAIL	Email address
E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title

Table 76. Attribute types supported by SSLPEER (continued).

A two column table describing the attributes supported by the SSLPEER parameter

Attribute	Description
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zipcode
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

WebSphere MQ accepts only uppercase letters for the attribute types.

If any of the unsupported attribute types are specified in the SSLPEER string, an error is output either when the attribute is defined, or at run time. When the error is output depends on which platform you are running on. An error implies that the SSLPEER string does not match the distinguished name of the flowed certificate.

If the distinguished name of the flowed certificate contains multiple organizational unit (OU) attributes, and SSLPEER specifies that these attributes are to be compared, they must be defined in descending hierarchical order. For example, if the distinguished name of the flowed certificate contains the OUs `OU=Large Unit, OU=Medium Unit, OU=Small Unit`, specifying the following SSLPEER values works:

```
('OU=Large Unit,OU=Medium Unit')
('OU=*,OU=Medium Unit,OU=Small Unit')
('OU=*,OU=Medium Unit')
```

but specifying the following SSLPEER values fails:

```
('OU=Medium Unit,OU=Small Unit')
('OU=Large Unit,OU=Small Unit')
('OU=Medium Unit')
('OU=Small Unit, Medium Unit, Large Unit')
```

As indicated in these examples, attributes at the low end of the hierarchy can be omitted. For example, `('OU=Large Unit,OU=Medium Unit')` is equivalent to `('OU=Large Unit,OU=Medium Unit,OU=*')`

If two DNs are equal in all respects except for their domain component (DC)) values, almost the same matching rules apply as for OUs. The exception is that with DC values, the left-most DC is the lowest-level and most specific, and the comparison ordering differs accordingly.

Any or all the attribute values can be generic, either an asterisk `*` on its own, or a stem with initiating or trailing asterisks. Asterisks allow the SSLPEER to match any distinguished name value, or any value starting with the stem for that attribute. You can specify an asterisk at the beginning or end of any attribute value in the DN on the certificate. If you do so, you can still check for an exact match with SSLPEER. Specify `\*` to check for an exact match. For example, if you have an attribute of `CN='Test*'` in the DN of the certificate, you use the following command to check for an exact match:

```
SSLPEER('CN=Test\*')
```

The maximum length of the parameter is 1024 bytes on AIX, HP-UX, IBM i, Linux, Solaris, and Windows platforms, and 256 bytes on z/OS.

STATCHL
Controls the collection of statistics data for channels:

QMGR The value of the STATCHL parameter of the queue manager is inherited by the channel.

OFF Statistics data collection is turned off for this channel.

LOW If the value of the STATCHL parameter of the queue manager is not NONE, statistics data collection is turned on. Data is collected at a low rate for this channel.

MEDIUM If the value of the STATCHL parameter of the queue manager is not NONE, statistics data collection is turned on. Data is collected at a medium rate for this channel.

HIGH If the value of the STATCHL parameter of the queue manager is not NONE, statistics data collection is turned on. Data is collected at a high rate for this channel.

Changes to this parameter take effect only on channels started after the change occurs.

For cluster channels, the value of this parameter is not replicated in the repository and used in the auto-definition of CLUSSDR channels. For auto-defined CLUSSDR channels, the value of this parameter is taken from the attribute STATACLS of the queue manager. This value might then be overridden in the channel auto-definition exit.

This parameter is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

TPNAME(*string*)
LU 6.2 transaction program name (maximum length 64 characters).

This parameter is valid only for channels with a transport type (TRPTYPE) of LU62.

Set this parameter to the SNA transaction program name, unless the CONNAME contains a side-object name in which case set it to blanks. The actual name is taken instead from the CPI-C Communications Side Object, or the APPC side information data set.

On Windows SNA Server, and in the side object on z/OS, the TPNAME is wrapped to uppercase.

This parameter is not valid for channels with a channel type (CHLTYPE) of RCVR.

TRPTYPE
Transport type to be used.

On AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS, this parameter is optional because, if you do not enter a value, the value specified in the SYSTEM.DEF.*channel-type* definition is used. If the channel is initiated from the other end, no check is made that the correct transport type is specified. On z/OS, if the SYSTEM.DEF.*channel-type* definition does not exist, the default is LU62.

This parameter is required on all other platforms.

LU62 SNA LU 6.2

NETBIOS
NetBIOS (supported only on Windows, and DOS; it also applies to z/OS for defining CLNTCONN channels that connect to servers on the platforms supporting NetBIOS)

SPX Sequenced packet exchange (supported only on Windows, and DOS; it also applies to z/OS for defining CLNTCONN channels that connect to servers on the platforms supporting SPX)

TCP Transmission Control Protocol - part of the TCP/IP protocol suite

USECLTID
Decide whether you want to use the IBM WebSphere MQ Telemetry client ID for the new connection as the WebSphere MQ user ID for that connection. If this property is specified, the user name supplied by the client is ignored.

USEDLQ

Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

NO Messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the NPMSPEED setting.

YES When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used, else the behavior is as for NO. YES is the default value.

USERID(*string*)

Task user identifier. The maximum length is 12 characters.

This parameter is used by the message channel agent when attempting to initiate a secure LU 6.2 session with a remote message channel agent.

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR, SVR, RQSTR, CLNTCONN, or CLUSSDR. On z/OS, it is supported only for CLNTCONN channels.

Although the maximum length of the parameter is 12 characters, only the first 10 characters are used.

On the receiving end, if passwords are encrypted and the LU 6.2 software is using a different encryption method, the channel fails to start. The error is diagnosed as invalid security details. You can avoid invalid security details by modifying the receiving SNA configuration to either:

- Turn off password substitution, or
- Define a security user ID and password.

XMITQ(*string*)

Transmission queue name.

The name of the queue from which messages are retrieved. See Rules for naming IBM WebSphere MQ objects .

This parameter is valid only for channels with a channel type (CHLTYPE) of SDR or SVR. For these channel types, this parameter is required.

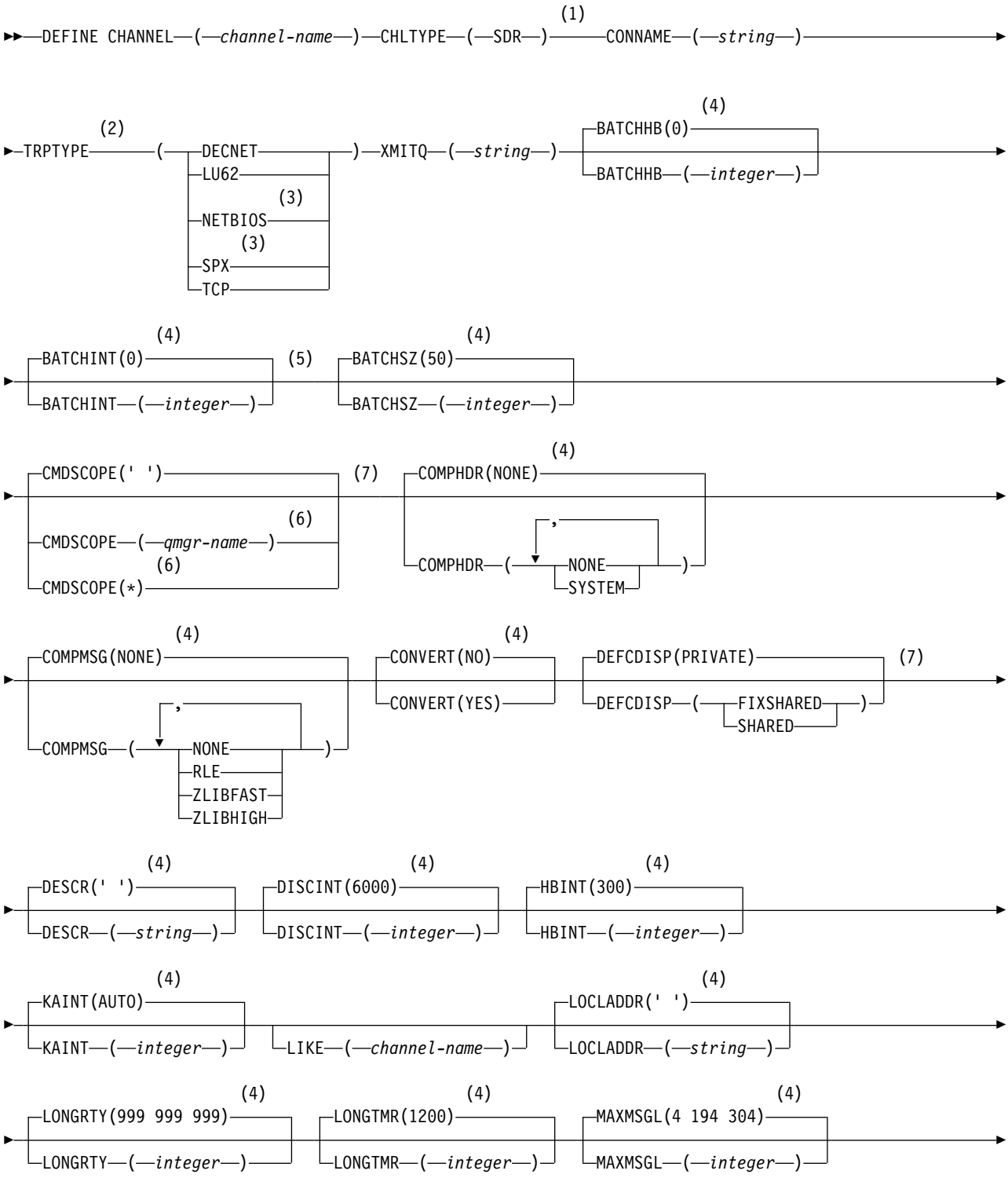
There is a separate syntax diagram for each type of channel:

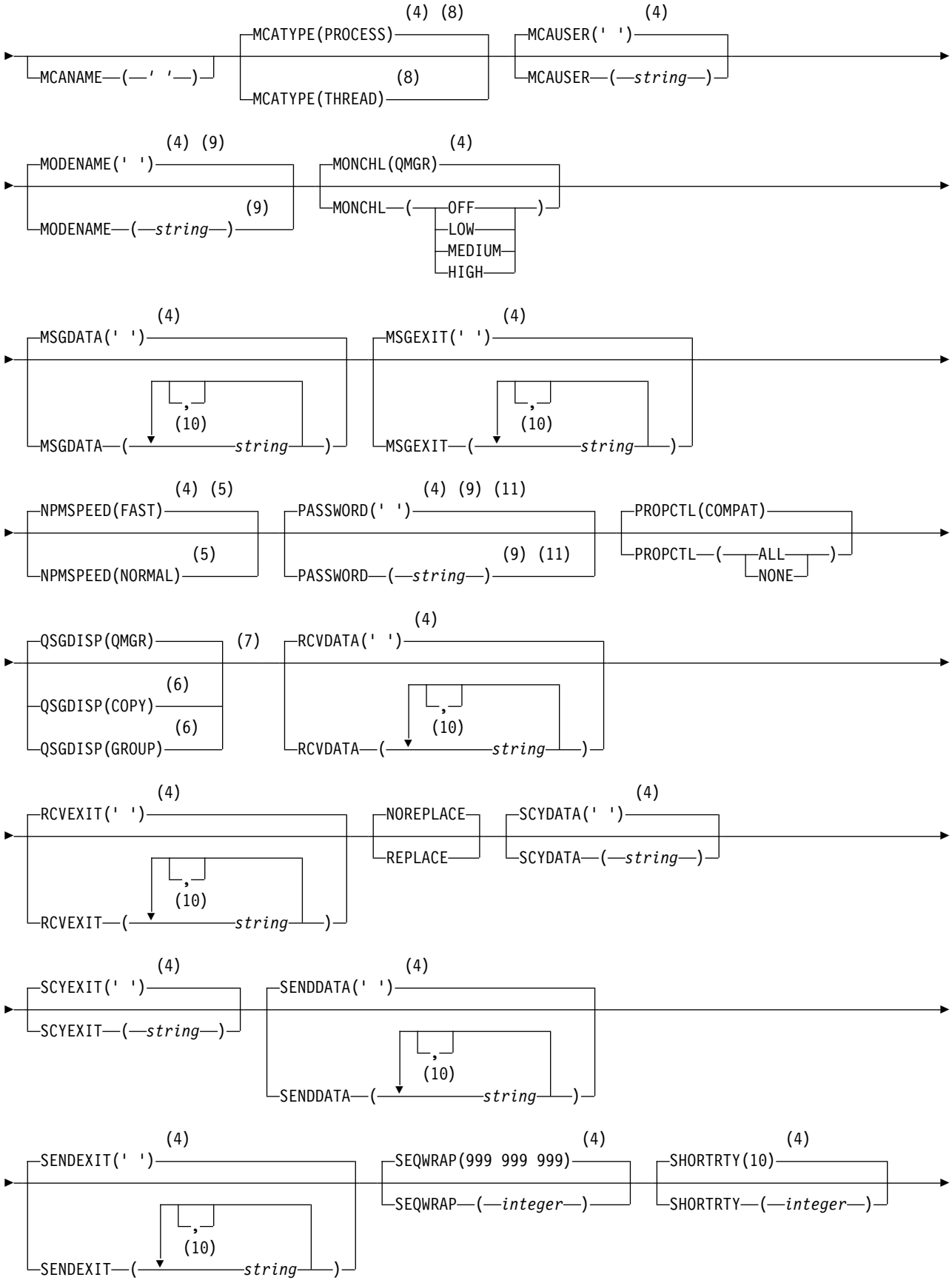
- "Sender channel" on page 471
- "Server channel" on page 473
- "Receiver channel" on page 476
- "Requester channel" on page 479
- "Client-connection channel" on page 481
- "Server-connection channel" on page 483
- "Cluster-sender channel" on page 484
- "Cluster-receiver channel" on page 487
- "DEFINE CHANNEL (MQTT)" on page 489

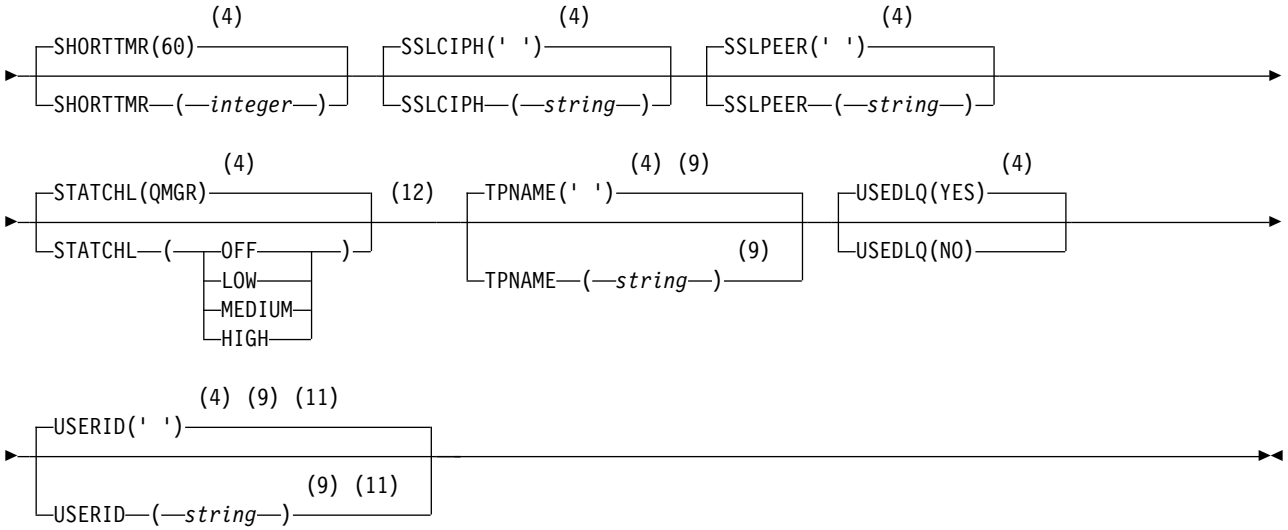
Sender channel:

Syntax diagram for a sender channel when using the DEFINE CHANNEL command.

DEFINE CHANNEL




MCANAME—(—' '—)
(4) (8)
MCATYPE(PROCESS)
(8)
MCATYPE(THREAD)
(4)
MCAUSER(' ')
MCAUSER—(—string—)
(4) (9)
MODENAME(' ')
(9)
MODENAME—(—string—)
(4)
MONCHL(QMGR)
MONCHL—(—OFF—)
LOW
MEDIUM
HIGH
(4)
MSGDATA(' ')
,
(10)
MSGDATA—(—string—)
(4)
MSGEXIT(' ')
,
(10)
MSGEXIT—(—string—)
(4) (5)
NPMSPEED(FAST)
(5)
NPMSPEED(NORMAL)
(4) (9) (11)
PASSWORD(' ')
(9) (11)
PASSWORD—(—string—)
PROPCTL(COMPAT)
PROPCTL—(—ALL—)
NONE
QSGDISP(QMGR)
(7)
(6)
QSGDISP(COPY)
(6)
QSGDISP(GROUP)
(4)
RCVDATA(' ')
,
(10)
RCVDATA—(—string—)
(4)
RCVEXIT(' ')
,
(10)
RCVEXIT—(—string—)
NOREPLACE
REPLACE
(4)
SCYDATA(' ')
SCYDATA—(—string—)
(4)
SCYEXIT(' ')
SCYEXIT—(—string—)
(4)
SENDDATA(' ')
,
(10)
SENDDATA—(—string—)
(4)
SENDEXIT(' ')
,
(10)
SENDEXIT—(—string—)
(4)
SEQWRAP(999 999 999)
SEQWRAP—(—integer—)
(4)
SHORTRTY(10)
SHORTRTY—(—integer—)



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 This is not mandatory on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

3 Valid only on Windows.

4 This is the default supplied with WebSphere MQ, but your installation might have changed it.

5 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

6 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

7 Valid only on z/OS.

8 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

9 Valid only if TRPTYPE is LU62.

10 You can specify more than one value only on AIX, HP-UX, Linux, IBM i, z/OS, Solaris, and Windows.

11 Not valid on z/OS.

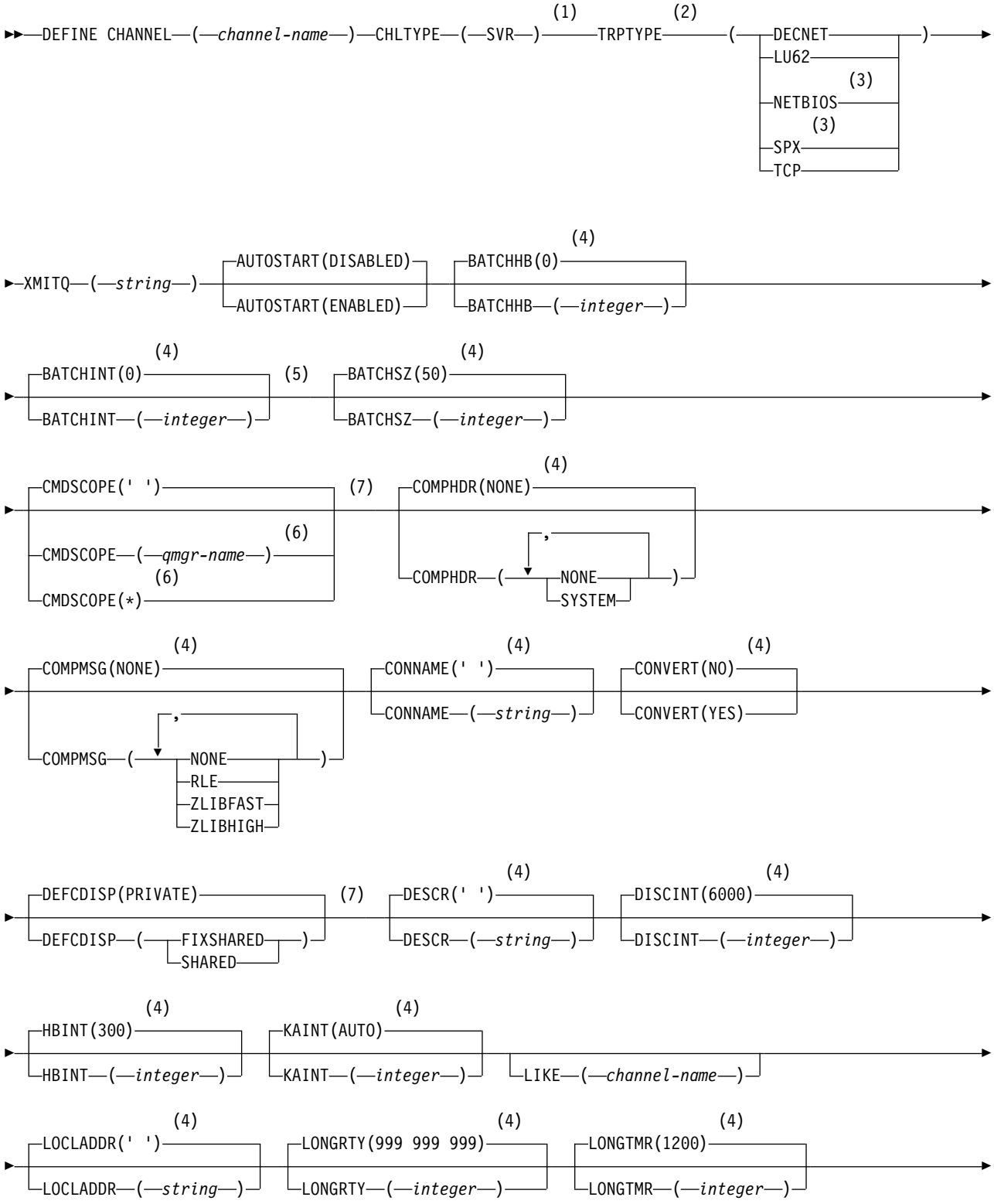
12 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

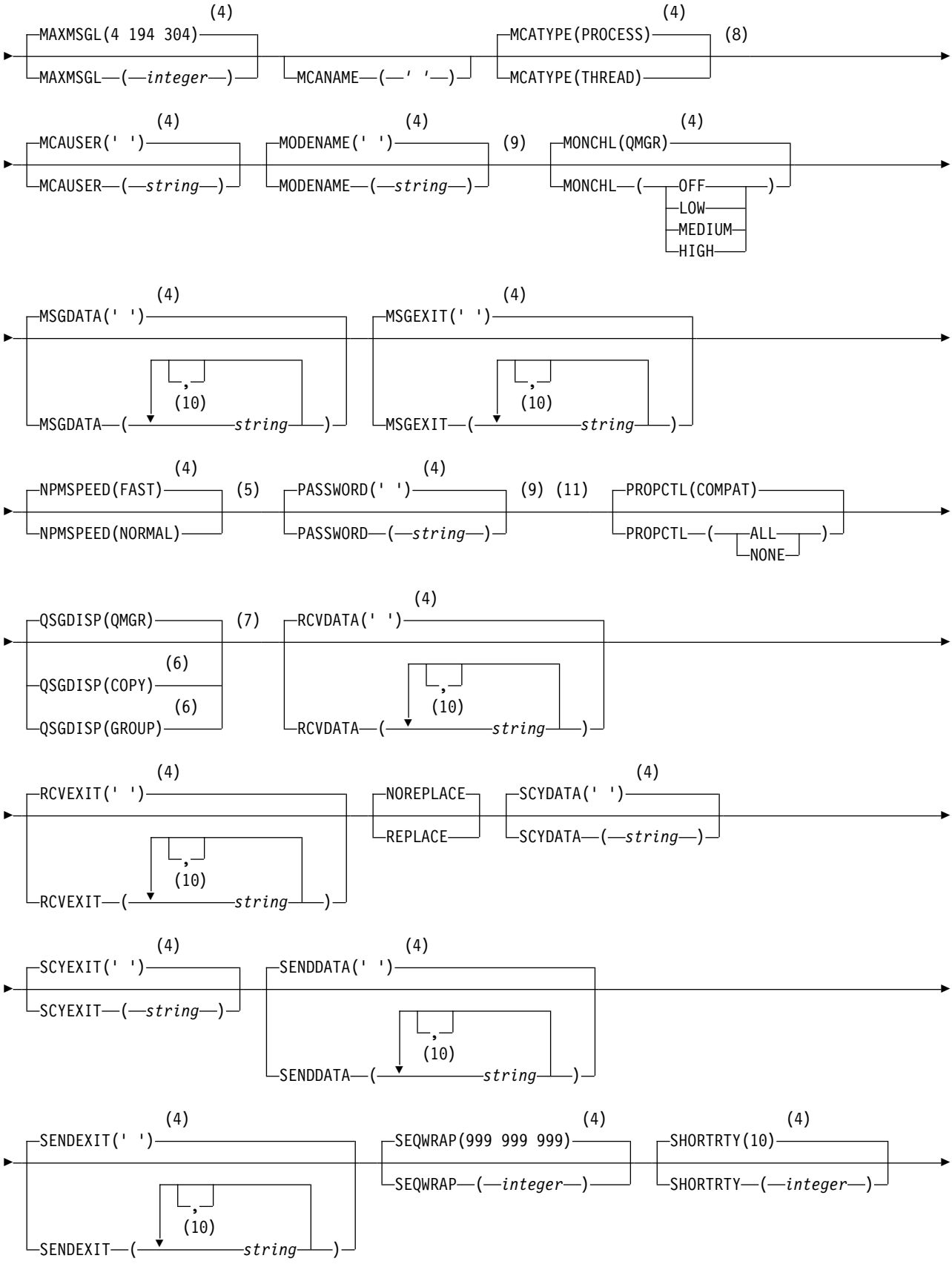
The parameters are described in “DEFINE CHANNEL” on page 437.

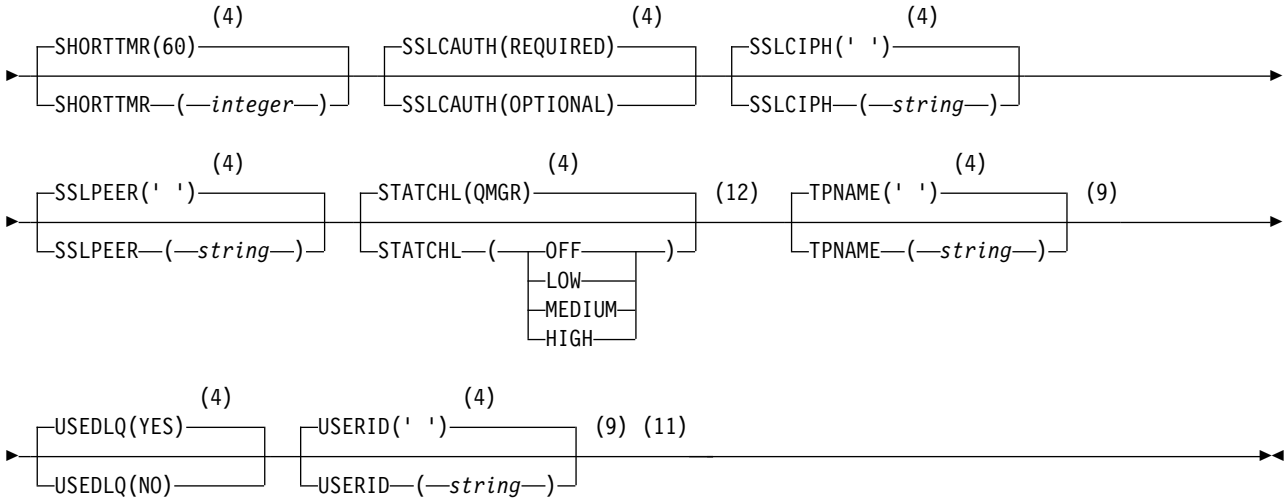
Server channel:

Syntax diagram for a server channel when using the DEFINE CHANNEL command.

DEFINE CHANNEL




(4)
MAXMSGL(4 194 304)
MAXMSGL—(—integer—)
MCANAME—(—' '—)
(4)
MCATYPE(PROCESS)
MCATYPE(THREAD)
(8)
(4)
MCAUSER(' ')
MCAUSER—(—string—)
(4)
MODENAME(' ')
MODENAME—(—string—)
(9)
(4)
MONCHL(QMGR)
MONCHL—(—OFF—)
LOW
MEDIUM
HIGH
(4)
MSGDATA(' ')
MSGDATA—(—,—(10)—string—)
(4)
MSGEXIT(' ')
MSGEXIT—(—,—(10)—string—)
(4)
NPMSPEED(FAST)
NPMSPEED(NORMAL)
(5)
(4)
PASSWORD(' ')
PASSWORD—(—string—)
(9) (11)
PROPCTL(COMPAT)
PROPCTL—(—ALL—)
NONE
QSGDISP(QMGR)
(7)
(6)
QSGDISP(COPY)
(6)
QSGDISP(GROUP)
(4)
RCVDATA(' ')
RCVDATA—(—,—(10)—string—)
(4)
RCVEXIT(' ')
RCVEXIT—(—,—(10)—string—)
NOREPLACE
REPLACE
(4)
SCYDATA(' ')
SCYDATA—(—string—)
(4)
SCYEXIT(' ')
SCYEXIT—(—string—)
(4)
SENDDATA(' ')
SENDDATA—(—,—(10)—string—)
(4)
SENDEXIT(' ')
SENDEXIT—(—,—(10)—string—)
(4)
SEQWRAP(999 999 999)
SEQWRAP—(—integer—)
(4)
SHORTRTY(10)
SHORTRTY—(—integer—)



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 This is not mandatory on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

3 Valid only on Windows.

4 This is the default supplied with WebSphere MQ, but your installation might have changed it.

5 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

6 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

7 Valid only on z/OS.

8 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

9 Valid only if TRPTYPE is LU62.

10 You can specify more than one value only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

11 Not valid on z/OS.

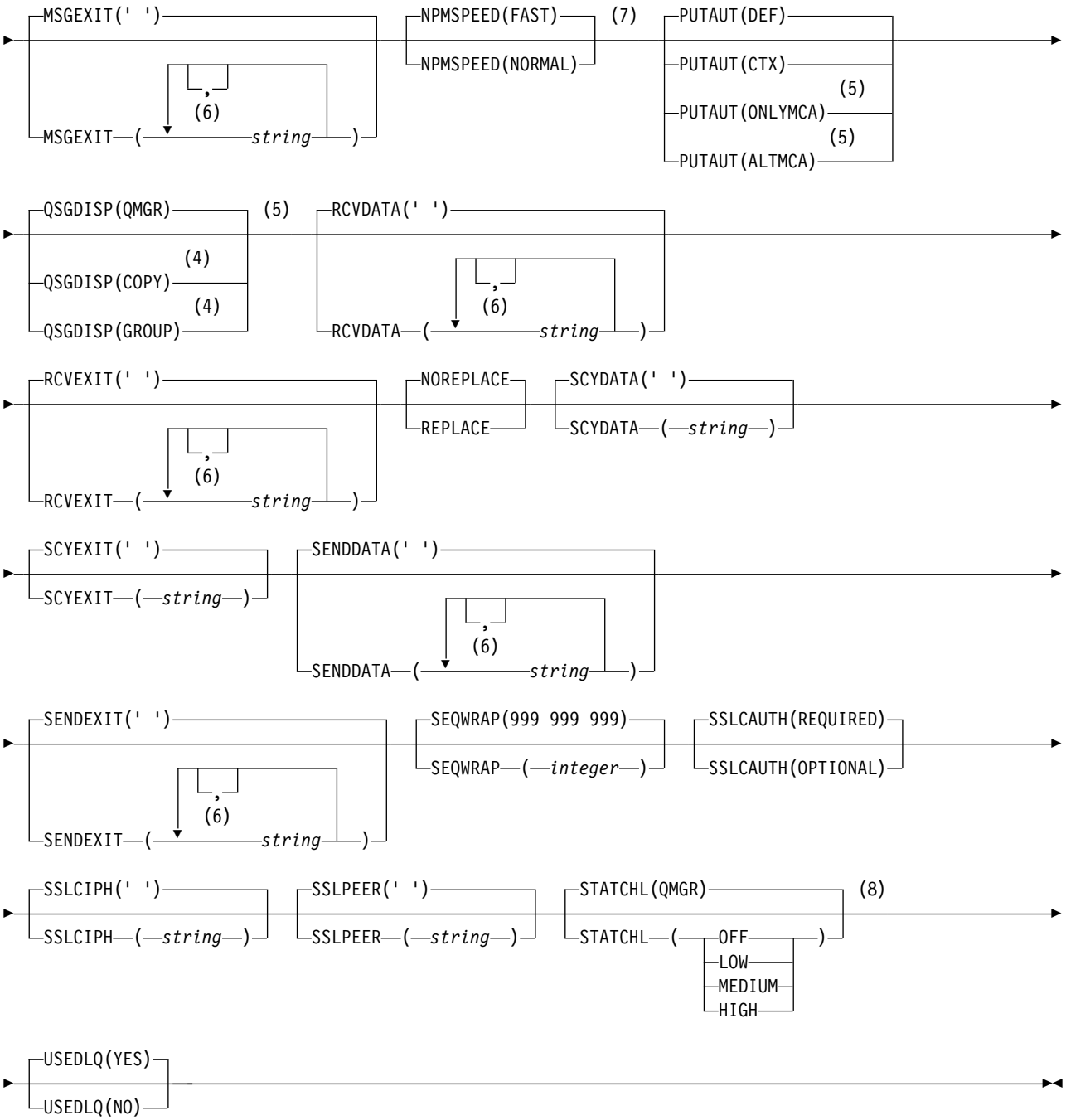
12 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

The parameters are described in “DEFINE CHANNEL” on page 437.

Receiver channel:

Syntax diagram for a receiver channel when using the DEFINE CHANNEL command.

DEFINE CHANNEL



Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 This is not mandatory on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

3 Valid only on Windows.

4 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

5 Valid only on z/OS.

6 You can specify more than one value only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

7 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

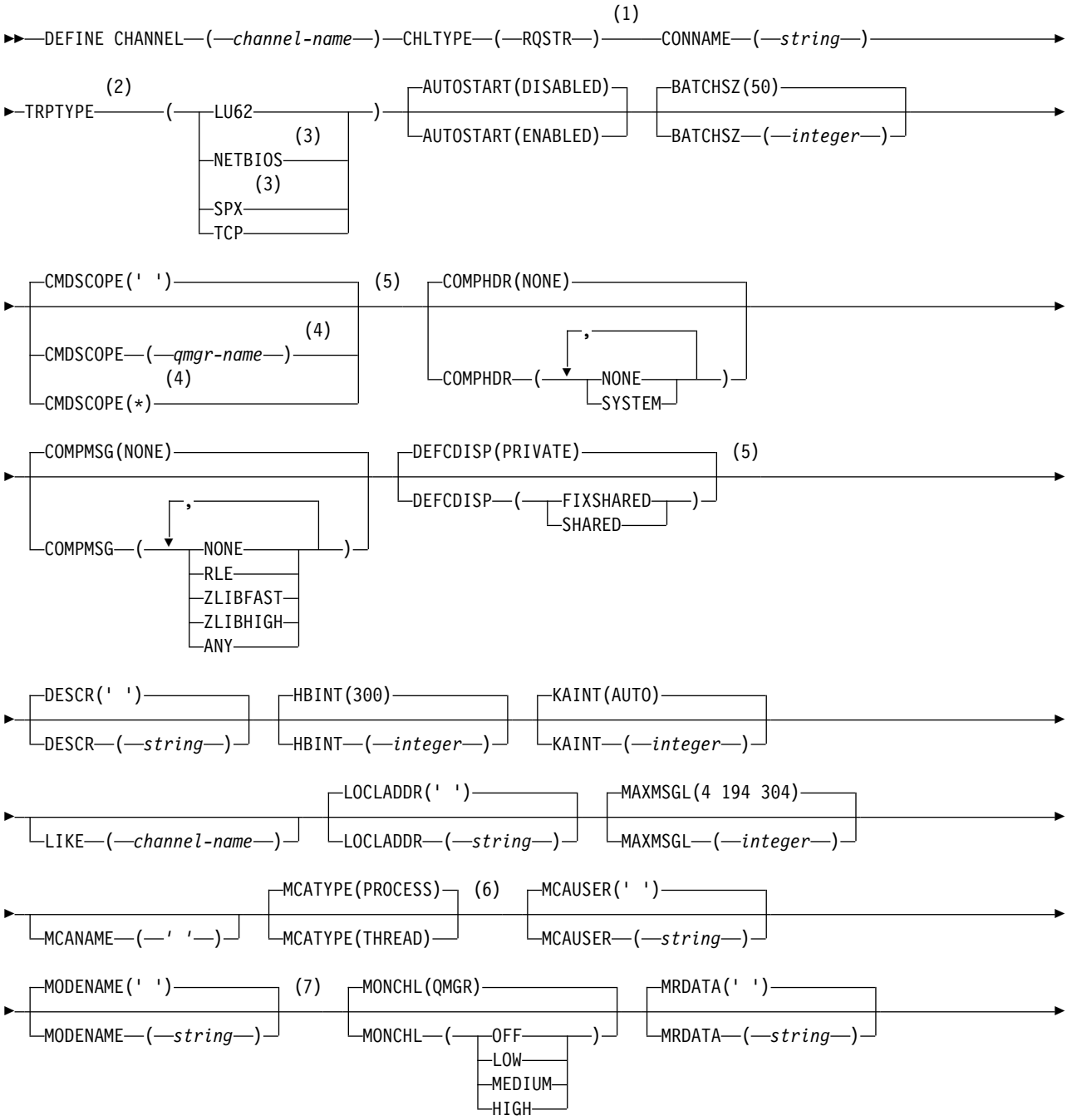
8 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

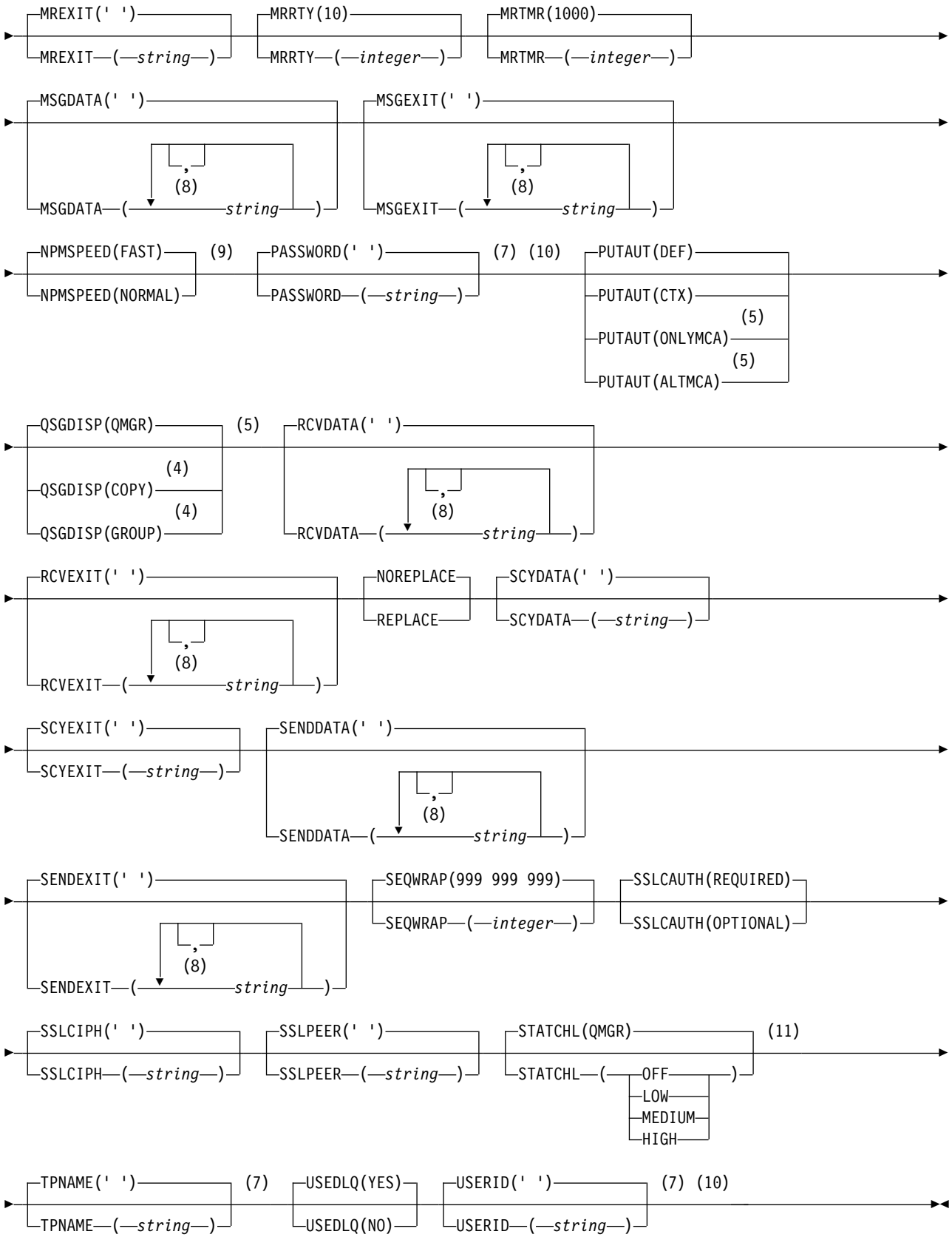
The parameters are described in "DEFINE CHANNEL" on page 437.

Requester channel:

Syntax diagram for a requester channel when using the DEFINE CHANNEL command.

DEFINE CHANNEL





Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 This is not mandatory on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

3 Valid only on Windows.

4 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

5 Valid only on z/OS.

6 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

7 Valid only if TRPTYPE is LU62.

8 You can specify more than one value only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

9 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

10 Not valid on z/OS.

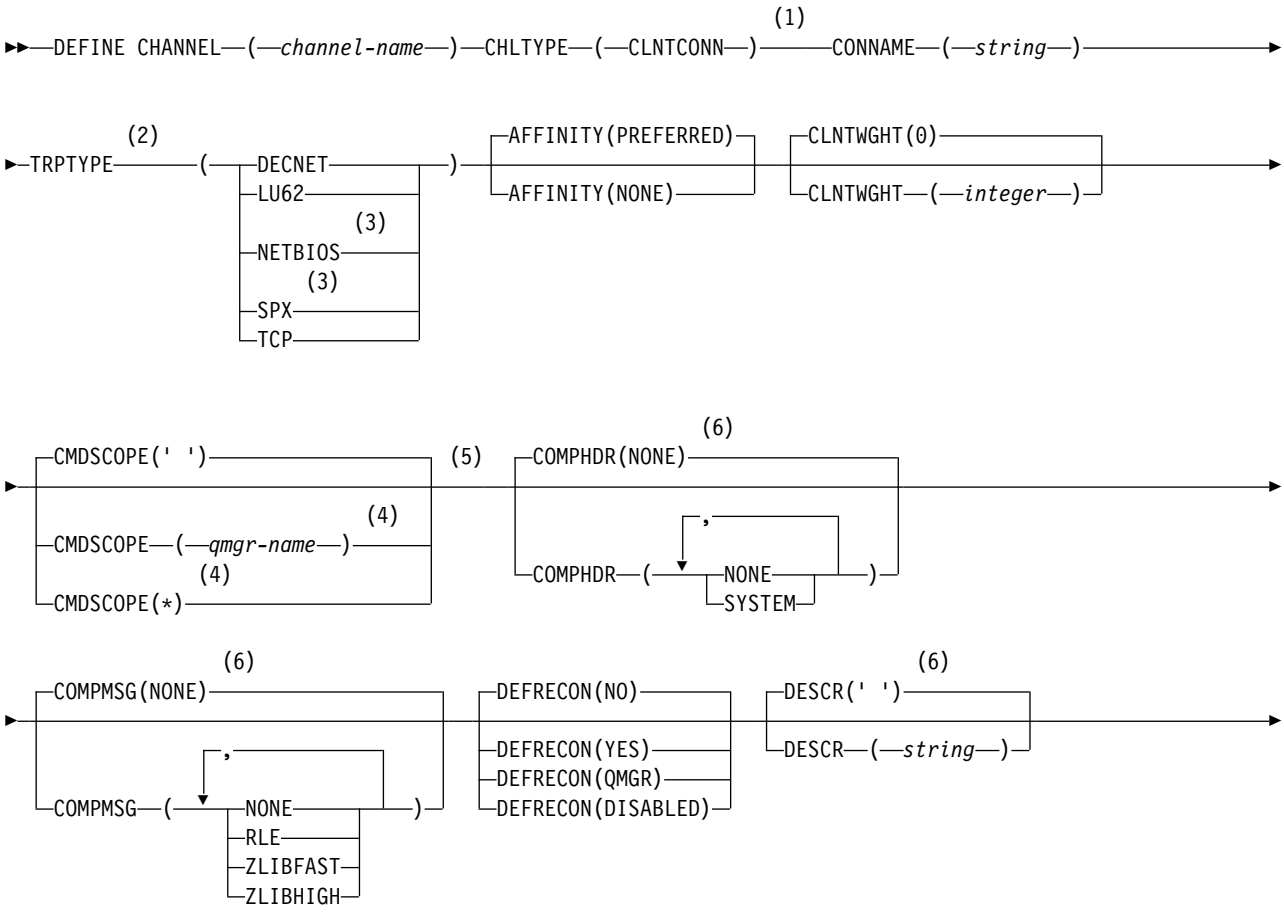
11 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

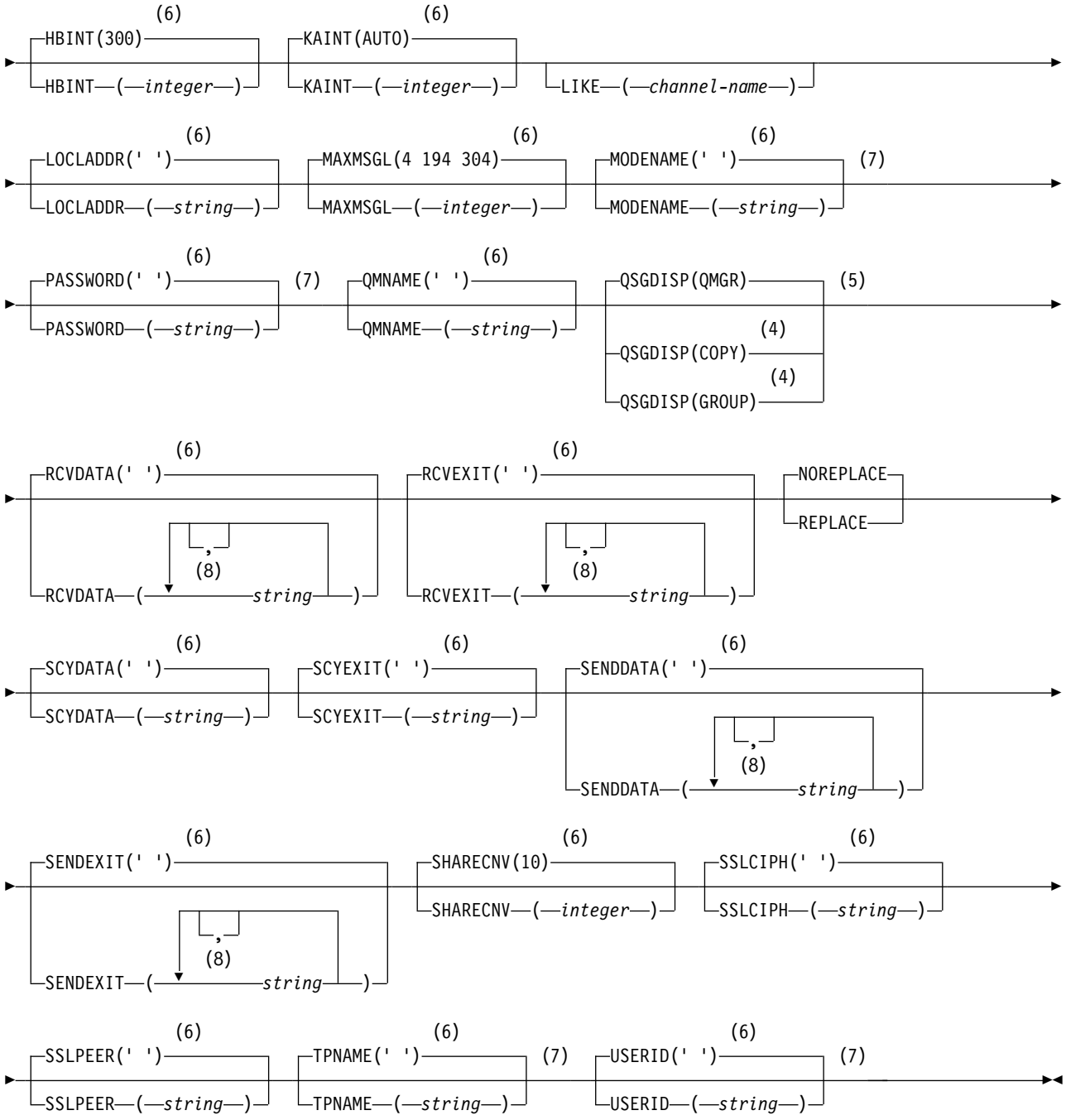
The parameters are described in "DEFINE CHANNEL" on page 437.

Client-connection channel:

Syntax diagram for a client-connection channel when using the DEFINE CHANNEL command.

DEFINE CHANNEL





Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 This is not mandatory on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

3 Valid only for clients to be run on DOS or Windows.

4 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

5 Valid only on z/OS.

6 This is the default supplied with WebSphere MQ, but your installation might have changed it.

7 Valid only if TRPTYPE is LU62.

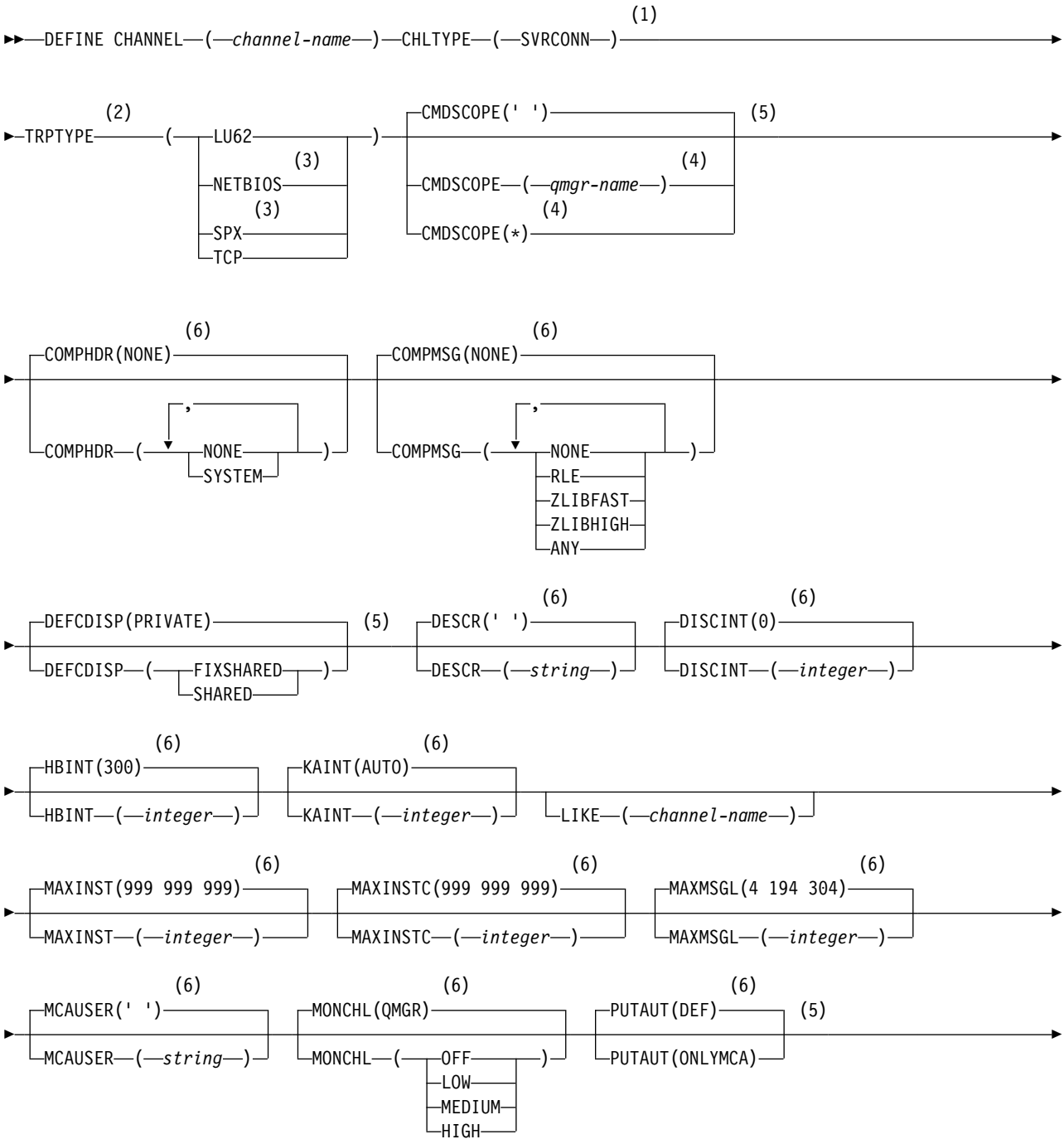
8 You can specify more than one value only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

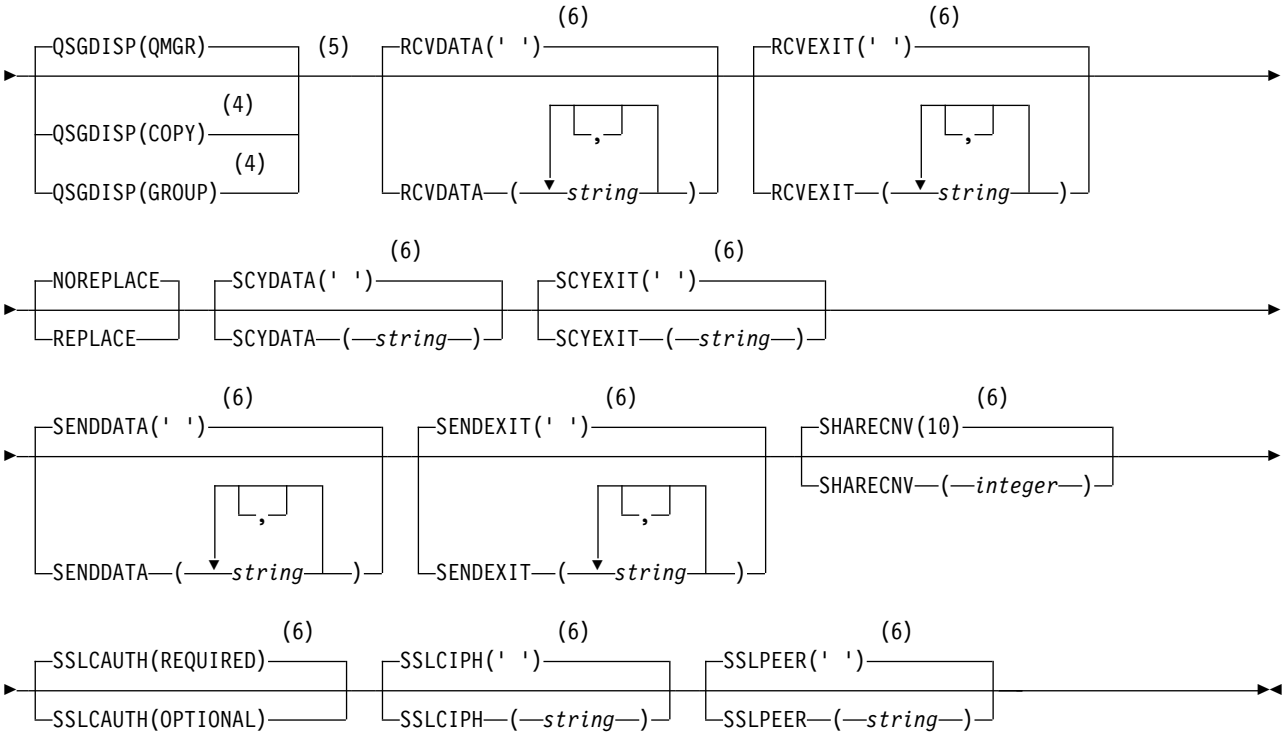
The parameters are described in "DEFINE CHANNEL" on page 437.

Server-connection channel:

Syntax diagram for a server-connection channel when using the DEFINE CHANNEL command.

DEFINE CHANNEL





Notes:

1 This parameter must follow immediately after the channel name except on z/OS.

2 This is not mandatory.

3 Valid only for clients to be run on Windows.

4 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

5 Valid only on z/OS.

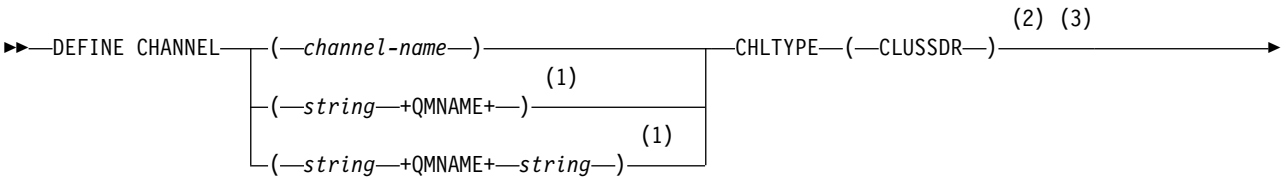
6 This is the default supplied with WebSphere MQ, but your installation might have changed it.

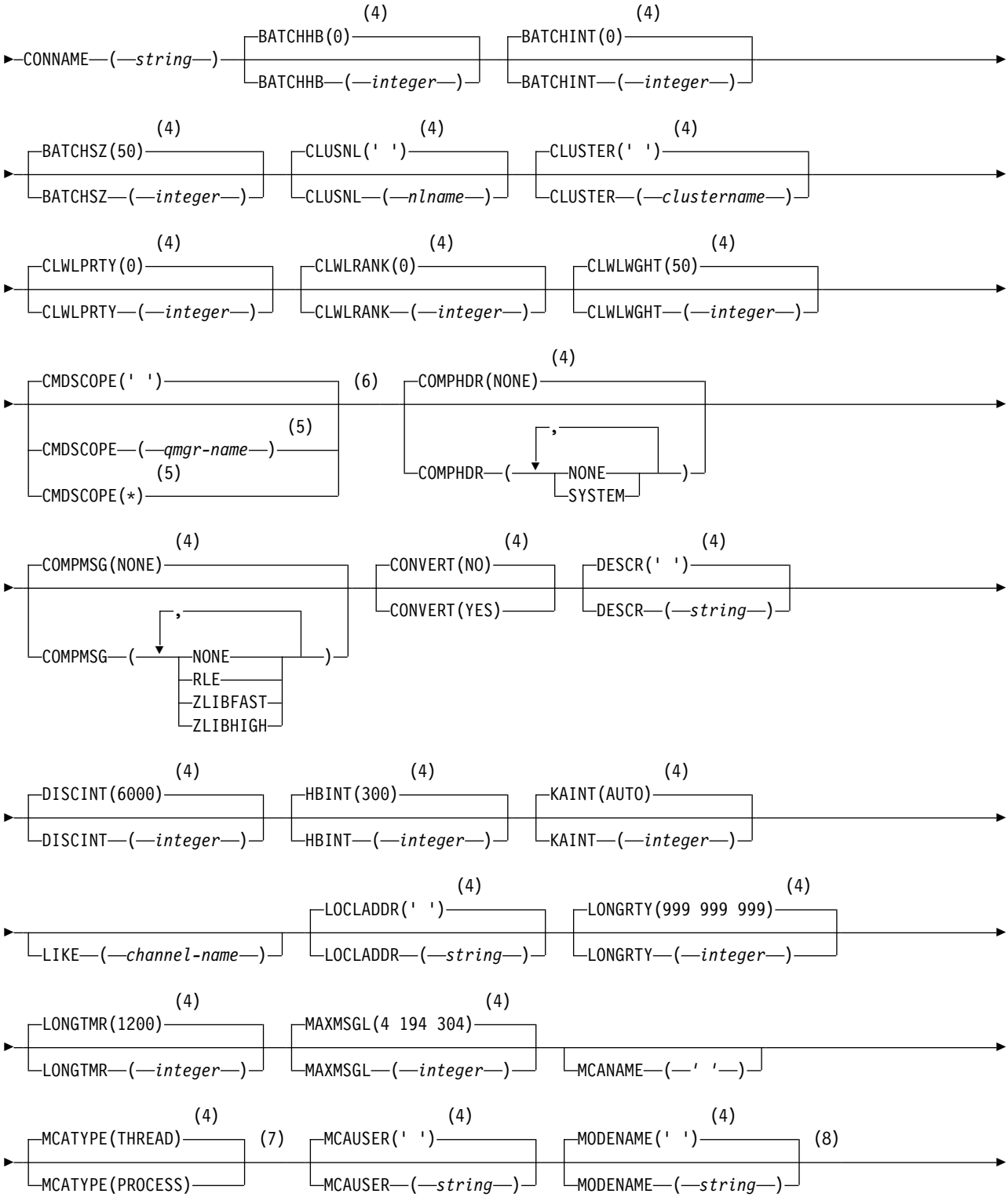
The parameters are described in "DEFINE CHANNEL" on page 437.

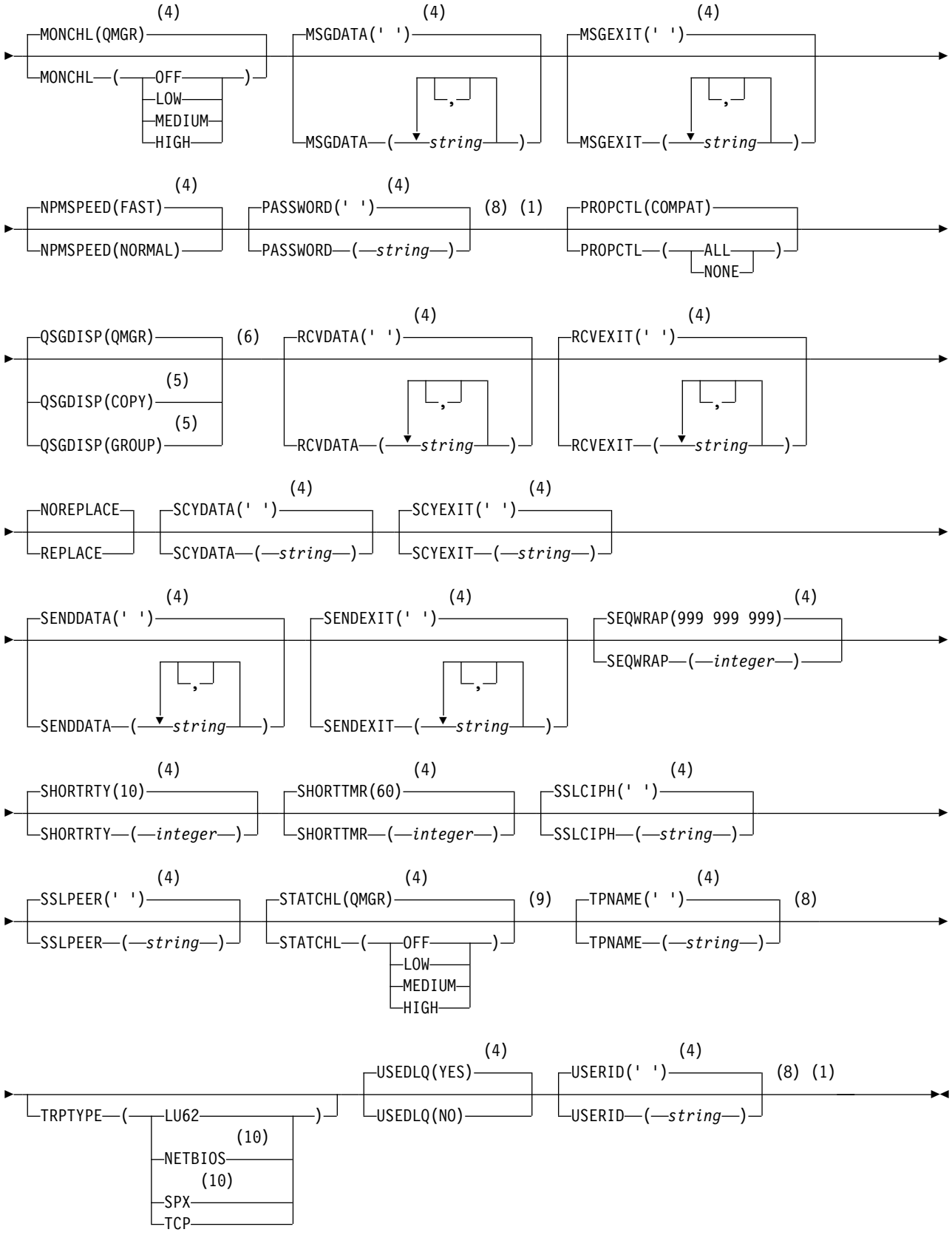
Cluster-sender channel:

Syntax diagram for a cluster-sender channel when using the DEFINE CHANNEL command.

DEFINE CHANNEL




CONNAME(string)
BATCHHB(0) (4)
BATCHHB(integer)
BATCHINT(0) (4)
BATCHINT(integer)
BATCHSZ(50) (4)
BATCHSZ(integer)
CLUSNL(' ') (4)
CLUSNL(nlname)
CLUSTER(' ') (4)
CLUSTER(clustername)
CLWLPRTY(0) (4)
CLWLPRTY(integer)
CLWLRANK(0) (4)
CLWLRANK(integer)
CLWLWGHT(50) (4)
CLWLWGHT(integer)
CMDSCOPE(' ')
CMDSCOPE(qmgr-name) (5)
CMDSCOPE(*) (5)
(6)
COMPHDR(NONE) (4)
COMPHDR(NONE,SYSTEM)
COMPMSG(NONE) (4)
COMPMSG(NONE,RLE,ZLIBFAST,ZLIBHIGH)
CONVERT(NO) (4)
CONVERT(YES)
DESCR(' ') (4)
DESCR(string)
DISCINT(6000) (4)
DISCINT(integer)
HBINT(300) (4)
HBINT(integer)
KAINT(AUTO) (4)
KAINT(integer)
LIKE(channel-name)
LOCLADDR(' ') (4)
LOCLADDR(string)
LONGRTY(999 999 999) (4)
LONGRTY(integer)
LONGTMR(1200) (4)
LONGTMR(integer)
MAXMSGL(4 194 304) (4)
MAXMSGL(integer)
MCANAME(' ')
MCATYPE(THREAD) (4)
MCATYPE(PROCESS)
(7)
MCAUSER(' ') (4)
MCAUSER(string)
MODENAME(' ') (4)
MODENAME(string)
(8)



Notes:

1 Not valid on z/OS.

2 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

3 This parameter must follow immediately after the channel name except on z/OS.

4 This is the default supplied with WebSphere MQ, but your installation might have changed it.

5 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

6 Valid only on z/OS.

7 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

8 Valid only if TRPTYPE is LU62.

9 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

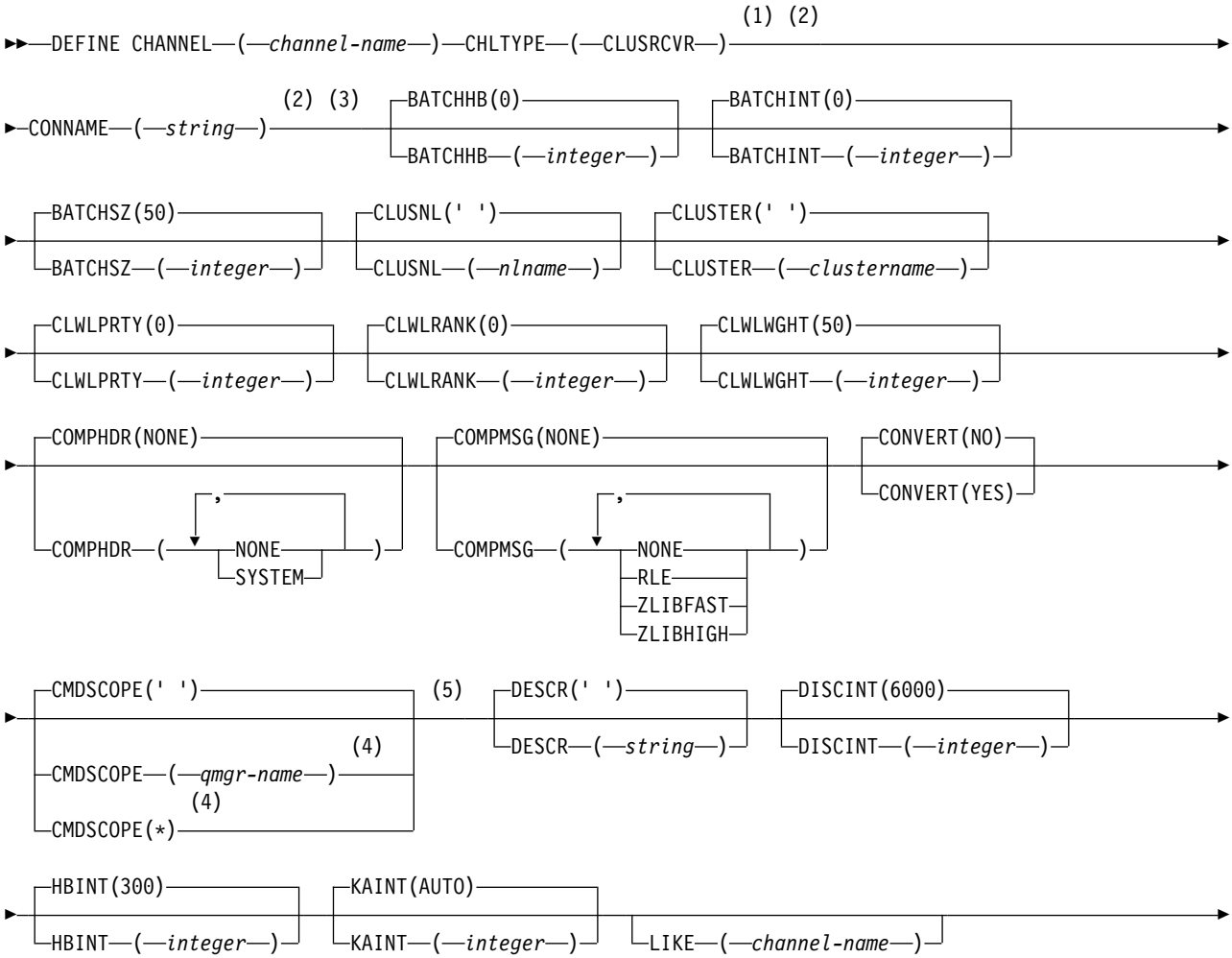
10 Valid only on Windows.

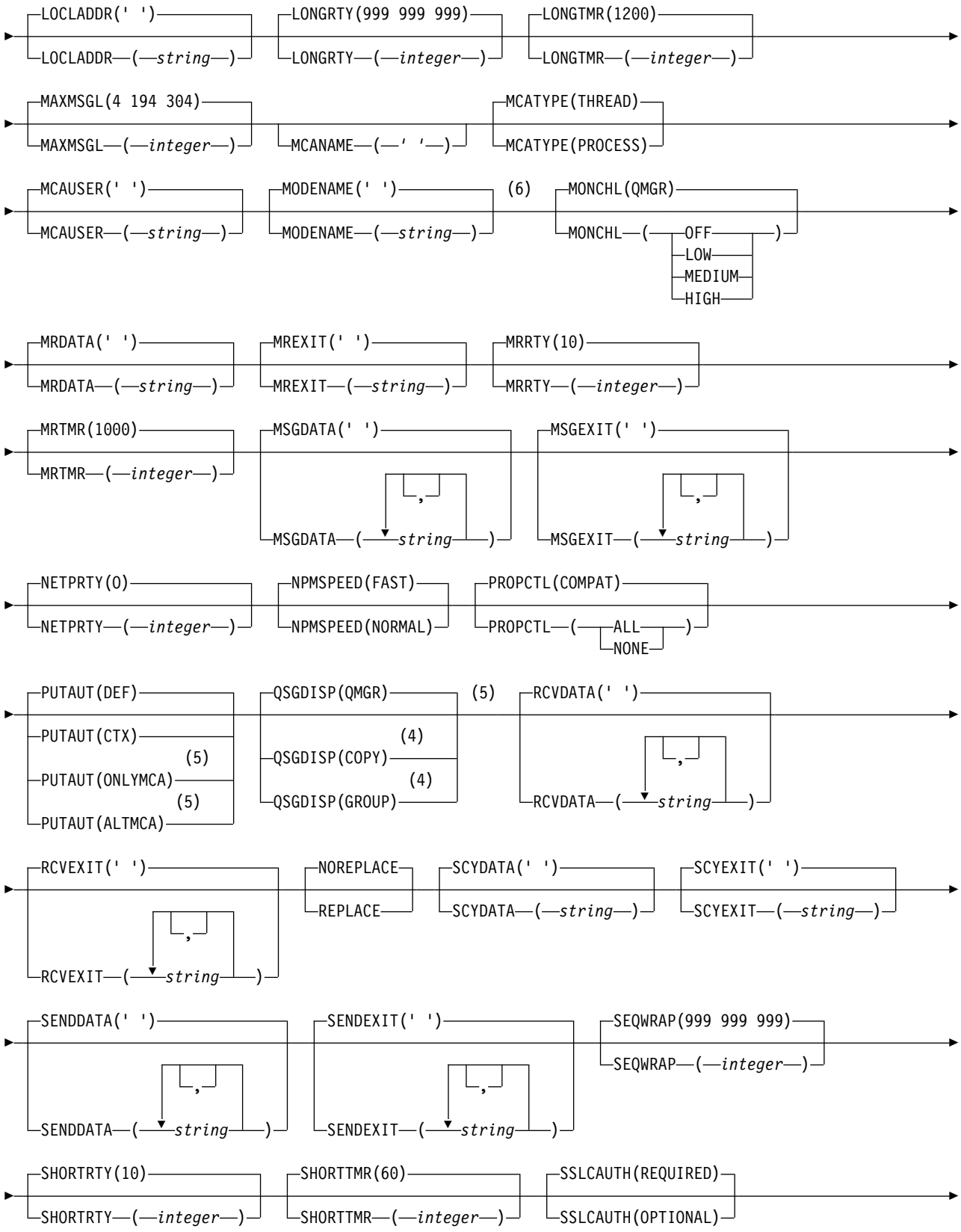
The parameters are described in "DEFINE CHANNEL" on page 437.

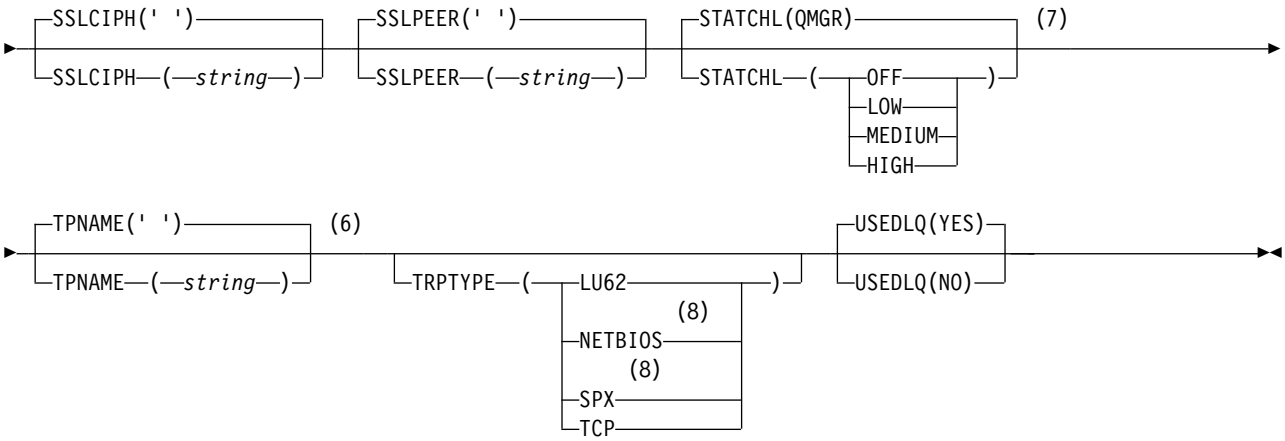
Cluster-receiver channel:

Syntax diagram for a cluster-receiver channel when using the DEFINE CHANNEL command.

DEFINE CHANNEL




LOCLADDR(' ')
LOCLADDR(string)
LONGRTY(999 999 999)
LONGRTY(integer)
LONGTMR(1200)
LONGTMR(integer)
MAXMSGL(4 194 304)
MAXMSGL(integer)
MCANAME(' ')
MCATYPE(THREAD)
MCATYPE(PROCESS)
MCAUSER(' ')
MCAUSER(string)
MODENAME(' ')
MODENAME(string)
(6)
MONCHL(QMGR)
MONCHL(OFF)
LOW
MEDIUM
HIGH
MRDATA(' ')
MRDATA(string)
MREXIT(' ')
MREXIT(string)
MRRTY(10)
MRRTY(integer)
MRTMR(1000)
MRTMR(integer)
MSGDATA(' ')
MSGDATA(string)
MSGEXIT(' ')
MSGEXIT(string)
NETPRTY(0)
NETPRTY(integer)
NPMSPEED(FAST)
NPMSPEED(NORMAL)
PROPCTL(COMPAT)
PROPCTL(ALL)
NONE
PUTAUT(DEF)
PUTAUT(CTX)
(5)
PUTAUT(ONLYMCA)
(5)
PUTAUT(ALTMCA)
QSGDISP(QMGR)
(5)
(4)
QSGDISP(COPY)
(4)
QSGDISP(GROUP)
RCVDATA(' ')
RCVDATA(string)
RCVEXIT(' ')
RCVEXIT(string)
NOREPLACE
REPLACE
SCYDATA(' ')
SCYDATA(string)
SCYEXIT(' ')
SCYEXIT(string)
SENDDATA(' ')
SENDDATA(string)
SENDEXIT(' ')
SENDEXIT(string)
SEQWRAP(999 999 999)
SEQWRAP(integer)
SHORTRTY(10)
SHORTRTY(integer)
SHORTTMR(60)
SHORTTMR(integer)
SSLCAUTH(REQUIRED)
SSLCAUTH(OPTIONAL)



Notes:

1 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

2 This parameter must follow immediately after the channel name except on z/OS.

3 This parameter is optional if TRPTYPE is TCP.

4 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

5 Valid only on z/OS.

6 Valid only if TRPTYPE is LU62.

7 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

8 Valid only on Windows.

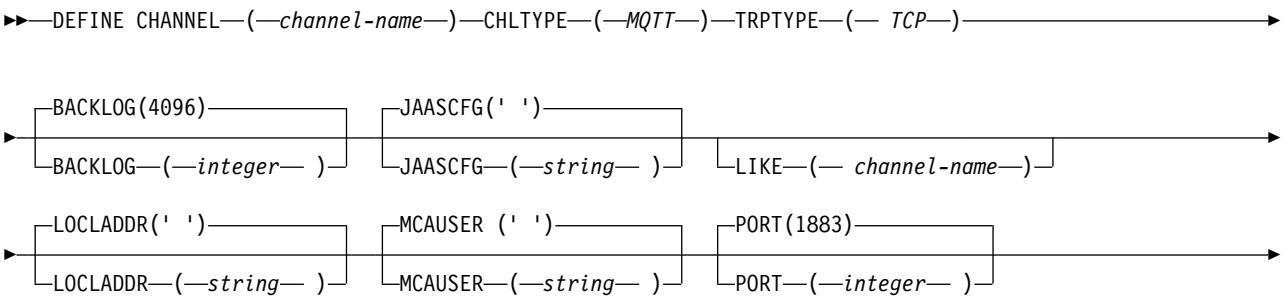
The parameters are described in "DEFINE CHANNEL" on page 437.

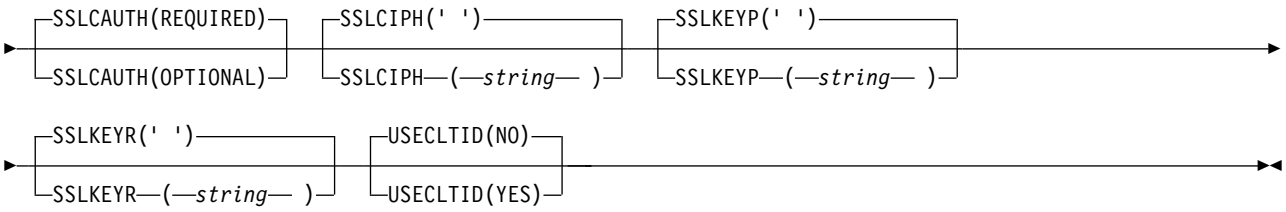
DEFINE CHANNEL (MQTT):

Syntax diagram for a telemetry channel when using the **DEFINE CHANNEL** command.

IBM i	UNIX and Linux	Windows	z/OS
	✓	✓	

Note: For the telemetry server, AIX is the only supported UNIX platform.





Usage notes

The telemetry (MQXR) service must be running when you issue this command. For instructions on how to start the telemetry (MQXR) service, see Configuring a queue manager for telemetry on Windows.

Parameter descriptions for DEFINE CHANNEL (MQTT)

(channel-name)
The name of the new channel definition.

The name must not be the same as any existing channel defined on this queue manager (unless REPLACE or ALTER is specified). On z/OS, client-connection channel names can duplicate others.

The maximum length of the string is 20 characters, and the string must contain only valid characters; see Rules for naming IBM WebSphere MQ objects.

BACKLOG(*integer***)**
The number of outstanding connection requests that the telemetry channel can support at any one time. When the backlog limit is reached, any further clients trying to connect will be refused connection until the current backlog is processed.

The value is in the range 0 - 999999999.

The default value is 4096.

CHLTYPE
Channel type.

MQTT Telemetry channel

JAASCFG(*string***)**
The name of a stanza in the JAAS configuration file.

LOCLADDR(*string***)**
LOCLADDR is the local communications address for the channel. Use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. LOCLADDR might be useful in recovery scenarios where a channel is restarted on a different TCP/IP stack. LOCLADDR is also useful to force a channel to use an IPv4 or IPv6 stack on a dual-stack system. You can also use LOCLADDR to force a channel to use a dual-mode stack on a single-stack system.

This parameter is valid only for channels with a transport type (TRPTYPE) of TCP. If TRPTYPE is not TCP, the data is ignored and no error message is issued.

The value is the optional IP address, and optional port or port range used for outbound TCP/IP communications. The format for this information is as follows:

Table 73 on page 452 shows how the LOCLADDR parameter can be used:
`LOCLADDR([ip-addr][(low-port[,high-port])]`*`[,[ip-addr][(low-port[,high-port])])`*

The maximum length of LOCLADDR, including multiple addresses, is `MQ_LOCAL_ADDRESS_LENGTH`.

If you omit LOCLADDR, a local address is automatically allocated.

Note, that you can set LOCLADDR for a C client using the Client Channel Definition Table (CCDT).

All the parameters are optional. Omitting the `ip-addr` part of the address is useful to enable the configuration of a fixed port number for an IP firewall. Omitting the port number is useful to select a particular network adapter without having the identify a unique local port number. The TCP/IP stack generates a unique port number.

Specify *`[,[ip-addr][(low-port[,high-port])]]`* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *`[,[ip-addr][(low-port[,high-port])]]`* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

`ip-addr`
`ip-addr` is specified in one of three forms:

IPv4 dotted decimal
For example `192.0.2.1`

IPv6 hexadecimal notation
For example `2001:DB8:0:0:0:0:0:0`

Alphanumeric host name form
For example `WWW.EXAMPLE.COM`

`low-port and high-port`
`low-port` and `high-port` are port numbers enclosed in parentheses.

Table 77. Examples of how the LOCLADDR parameter can be used

LOCLADDR	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98, 9.20.4.99	Channel binds to either IP address. The address might be two network adapters on one server, or a different network adapter on two different servers in a multi-instance configuration.
9.20.4.98(1000)	Channel binds to this address and port 1000 locally
9.20.4.98(1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to port in range 1000 - 2000 locally

This parameter is valid only for channels with a channel type (CHLTYPE) of `SDR`, `SVR`, `RQSTR`, `CLNTCONN`, `CLUSSDR`, `CLUSRCVR`, or `MQTT`.

On `CLUSSDR` channels, the IP address and port to which the outbound channel binds, is a combination of fields. It is a concatenation of the IP address, as defined in the LOCLADDR parameter, and the port range from the cluster cache. If there is no port range in the cache, the port range defined in the LOCLADDR parameter is used. This port range does not apply to z/OS.

Even though this parameter is similar in form to CONNAME, it must not be confused with it. The LOCLADDR parameter specifies the characteristics of the local communications, whereas the CONNAME parameter specifies how to reach a remote queue manager.

When a channel is started, the values specified for CONNAME and LOCLADDR determine the IP stack to be used for communication; see Table 3 and Local Address (LOCLADDR) .

If the TCP/IP stack for the local address is not installed or configured, the channel does not start and an exception message is generated. The message indicates that the connect() request specifies an interface address that is not known on the default IP stack. To direct the connect() request to the alternative stack, specify the **LOCLADDR** parameter in the channel definition as either an interface on the alternative stack, or a DNS host name. The same specification also works for listeners that might not use the default stack. To find the value to code for **LOCLADDR**, run the **NETSTAT HOME** command on the IP stacks that you want to use as alternatives.

For channels with a channel type (CHLTYPE) of MQTT the usage of this parameter is slightly different. Specifically, a telemetry channel (MQTT) **LOCLADDR** parameter expects only an IPv4 or IPv6 IP address, or a valid host name as a string. This string must not contain a port number or port range. If an IP address is entered, only the address format is validated. The IP address itself is not validated.

Table 78. How the IP stack to be used for communication is determined

Protocols supported	CONNAME	LOCLADDR	Action of channel
IPv4 only	IPv4 address[1]		Channel binds to IPv4 stack
	IPv6 address[2]		Channel fails to resolve CONNAME
	IPv4 and 6 host name[3]		Channel binds to IPv4 stack
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	Any address[4]	IPv6 address	Channel fails to resolve LOCLADDR
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv4 stack
IPv4 and IPv6	IPv4 address		Channel binds to IPv4 stack
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to stack determined by IPADDRV
	IPv4 address	IPv4 address	Channel binds to IPv4 stack
	IPv6 address	IPv4 address	Channel fails to resolve CONNAME
	IPv4 and 6 host name	IPv4 address	Channel binds to IPv4 stack
	IPv4 address	IPv6 address	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv6 address	Channel binds IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel binds to IPv4 stack
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to stack determined by IPADDRV

Table 78. How the IP stack to be used for communication is determined (continued)

Protocols supported	CONNAME	LOCLADDR	Action of channel
IPv6 only	IPv4 address		Channel maps CONNAME to IPv6[5]
	IPv6 address		Channel binds to IPv6 stack
	IPv4 and 6 host name		Channel binds to IPv6 stack
	Any address	IPv4 address	Channel fails to resolve LOCLADDR
	IPv4 address	IPv6 address	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv6 address	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv6 address	Channel binds to IPv6 stack
	IPv4 address	IPv4 and 6 host name	Channel maps CONNAME to IPv6[5]
	IPv6 address	IPv4 and 6 host name	Channel binds to IPv6 stack
	IPv4 and 6 host name	IPv4 and 6 host name	Channel binds to IPv6 stack

Notes:

1. IPv4 address. An IPv4 host name that resolves only to an IPv4 network address or a specific dotted notation IPv4 address, for example `1.2.3.4`. This note applies to all occurrences of 'IPv4 address' in this table.
2. IPv6 address. An IPv6 host name that resolves only to an IPv6 network address or a specific hexadecimal notation IPv6 address, for example `4321:54bc`. This note applies to all occurrences of 'IPv6 address' in this table.
3. IPv4 and 6 host name. A host name that resolves to both IPv4 and IPv6 network addresses. This note applies to all occurrences of 'IPv4 and 6 host name' in this table.
4. Any address. IPv4 address, IPv6 address, or IPv4 and 6 host name. This note applies to all occurrences of 'Any address' in this table.
5. Maps IPv4 CONNAME to IPv4 mapped IPv6 address. IPv6 stack implementations that do not support IPv4 mapped IPv6 addressing fail to resolve the CONNAME. Mapped addresses might require protocol translators in order to be used. The use of mapped addresses is not recommended.

MCAUSER(*string*)

Message channel agent user identifier.

This parameter interacts with PUTAUT , see the definition of that parameter for more information.

If it is nonblank, it is the user identifier that is to be used by the message channel agent for authorization to access WebSphere MQ resources, including (if PUTAUT is DEF) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

The default user identifier is derived from the user ID that started the receiving channel. The possible values are:

- On z/OS, the user ID assigned to the channel-initiator started task by the z/OS started-procedures table.
- For TCP/IP, other than z/OS , the user ID from the `inetd.conf` entry, or the user that started the listener.
- For SNA, other than z/OS , the user ID from the SNA server entry or, in the absence of this user ID the incoming attach request, or the user that started the listener.
- For NetBIOS or SPX, the user ID that started the listener.

The maximum length of the string is 64 characters on Windows and 12 characters on other platforms. On Windows, you can optionally qualify a user identifier with the domain name in the format `user@domain`.

This parameter is not valid for channels with a channel type (CHLTYPE) of `SDR`, `SVR`, `CLNTCONN`, `CLUSSDR`.

PORT(*integer***)**

The port number that the telemetry (MQXR) service accepts client connections on. The default port number for a telemetry channel is 1883; and the default port number for a telemetry channel secured using SSL is 8883. Specifying a port value of 0 causes MQTT to dynamically allocate an available port number.

SSLCAUTH

Defines whether WebSphere MQ requires a certificate from the SSL client. The initiating end of the channel acts as the SSL client, so this parameter applies to the end of the channel that receives the initiation flow, which acts as the SSL server.

This parameter is valid only for channels with a channel type (CHLTYPE) of RCVR, SVRCONN, CLUSRCVR, SVR, RQSTR, or MQTT.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

REQUIRED

WebSphere MQ requires and validates a certificate from the SSL client.

OPTIONAL

The peer SSL client system might still send a certificate. If it does, the contents of this certificate are validated as normal.

SSLCIPH*(string)*

When SSLCIPH is used with a telemetry channel, it means "SSL Cipher Suite". The SSL cipher suite is the one supported by the JVM that is running the telemetry (MQXR) service. If the SSLCIPH parameter is blank, no attempt is made to use SSL on the channel.

Here is an alphabetic list of the SSL cipher suites that are currently supported:

- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_WITH_AES_128_CBC_SHA
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_AES_128_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_RC4_128_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_AES_128_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_MD5
- SSL_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- SSL_KRB5_EXPORT_WITH_RC4_40_MD5
- SSL_KRB5_EXPORT_WITH_RC4_40_SHA
- SSL_KRB5_WITH_3DES_EDE_CBC_MD5
- SSL_KRB5_WITH_3DES_EDE_CBC_SHA
- SSL_KRB5_WITH_DES_CBC_MD5
- SSL_KRB5_WITH_DES_CBC_SHA

- SSL_KRB5_WITH_RC4_128_MD5
- SSL_KRB5_WITH_RC4_128_SHA
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_FIPS_WITH_AES_256_CBC_SHA256
- SSL_RSA_FIPS_WITH_DES_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_AES_128_CBC_SHA
- V7.5.0.2 SSL_RSA_WITH_AES_128_CBC_SHA256
- V7.5.0.2 SSL_RSA_WITH_AES_256_CBC_SHA256
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_NULL_SHA
- V7.5.0.2 SSL_RSA_WITH_NULL_SHA256
- SSL_RSA_WITH_RC4_128_MD5
- SSL_RSA_WITH_RC4_128_SHA

V7.5.0.2 If you plan to use SHA-2 cipher suites, see System requirements for using SHA-2 cipher suites with MQTT channels.

SSLKEYP(*string*)
The store for digital certificates and their associated private keys. If you do not specify a key file, SSL is not used.

SSLKEYR(*string*)
The password for the key repository. If no passphrase is entered, then unencrypted connections must be used.

USECLTID
Decide whether you want to use the MQTT client ID for the new connection as the IBM WebSphere MQ user ID for that connection. If this property is specified, the user name supplied by the client is ignored.

Related concepts:

../com.ibm.mq.adm.doc/q021331_.dita
The WebSphere MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. The channels are configured either as com.ibm.mq.MQTT.channel/PlainText or com.ibm.mq.MQTT.channel/SSL. SSL channels are configured with passphrase protected access to key files. If an SSL channel is defined with no passphrase or key file, the channel does not accept SSL connections.

../com.ibm.mq.adm.doc/q021341_.dita
The WebSphere MQ administrator configures telemetry channels at the server. Each channel is configured to accept a TCP/IP connection on a different port number. The channels are configured either as com.ibm.mq.MQTT.channel/PlainText or com.ibm.mq.MQTT.channel/SSL. SSL channels are configured with passphrase protected access to key files. If an SSL channel is defined with no passphrase or key file, the channel does not accept SSL connections.

../com.ibm.mq.sec.doc/q009950_.dita
Cryptographic security protocols must agree on the algorithms used by a secure connection. CipherSpecs and CipherSuites define specific combinations of algorithms.

Related reference:
"ALTER CHANNEL (MQTT)" on page 335
Syntax diagram for a telemetry channel when using the ALTER CHANNEL command. This is separate from the regular ALTER CHANNEL syntax diagram and parameter descriptions.

V 7.5.0.2 ../com.ibm.mq.tro.doc/q039371_.dita
For Java 6 from IBM, SR13 onwards, you can use SHA-2 cipher suites to secure your MQTT channels and client apps. However, SHA-2 cipher suites are not enabled by default until Java 7 from IBM, SR4 onwards, so in earlier versions you must specify the required suite. If you are running an MQTT client with your own JRE, you need to ensure that it supports the SHA-2 cipher suites. For your client apps to use SHA-2 cipher suites, the client must also set the SSL context to a value that supports Transport Layer Security (TLS) version 1.2.

DEFINE COMMINFO:

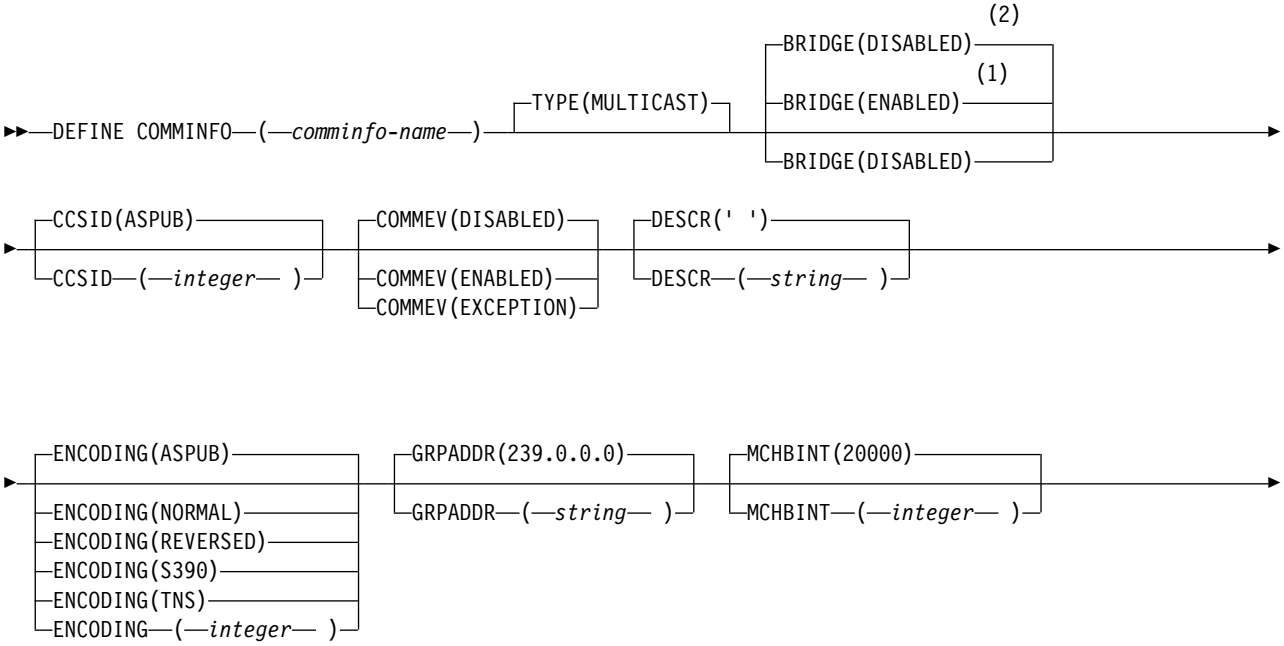
Use the MQSC command DEFINE COMMINFO to define a new communication information object. These objects contain the definitions required for Multicast messaging.

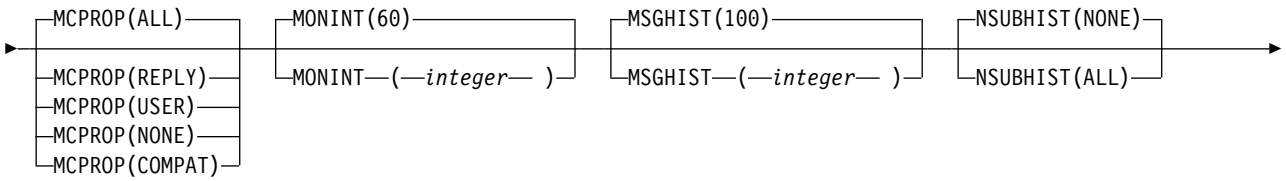
UNIX and Linux	Windows
✓	✓

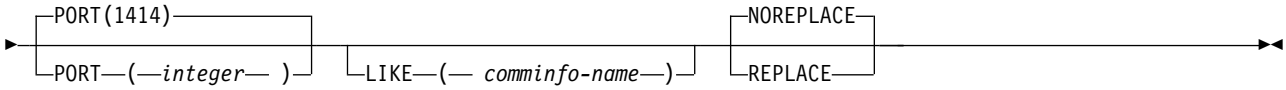
- Syntax diagram
- "Parameter descriptions for DEFINE COMMINFO" on page 497

Synonym: DEF COMMINFO

DEFINE COMMINFO







Notes:

1 Default for platforms other than IBM i.

2 Default for IBM i.

Parameter descriptions for DEFINE COMMINFO

(comminfo name)
Name of the communications information object. This is required.

The name must not be the same as any other communications information object name currently defined on this queue manager. See Rules for naming IBM WebSphere MQ objects .

TYPE The type of the communications information object. The only type supported is `MULTICAST`.

BRIDGE
Controls whether publications from applications not using Multicast are bridged to applications using Multicast. Bridging does not apply to topics that are marked as **MCAST(ONLY)** . As these topics can only be Multicast traffic, it is not applicable to bridge to the queue's publish/subscribe domain.

DISABLED
Publications from applications not using Multicast are not bridged to applications that do use Multicast. This is the default for IBM i.

ENABLED
Publications from applications not using Multicast are bridged to applications that do use Multicast. This is the default for platforms other than IBM i.

CCSID(*integer***)**
The coded character set identifier that messages are transmitted on. Specify a value in the range 1 through 65535.

The CCSID must specify a value that is defined for use on your platform, and use a character set that is appropriate to the platform. If you use this parameter to change the CCSID, applications that are running when the change is applied continue to use the original CCSID. Because of this, you must stop and restart all running applications before you continue. This includes the command server and channel programs. To do this, stop and restart the queue manager after making the change.

The default value is `ASPUB` which means that the coded character set is taken from the one that is supplied in the published message.

COMMEV
Controls whether event messages are generated for Multicast handles that are created using this COMMINFO object. Events will only be generated if they are enabled using the **MONINT** parameter.

DISABLED
Event messages are not generated for Multicast handles that are created using the COMMINFO object. This is the default value.

ENABLED
Event messages are generated for Multicast handles that are created using the COMMINFO object.

EXCEPTION
Event messages are written if the message reliability is below the reliability threshold The reliability threshold is set to 90 by default.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the communication information object when an operator issues the DISPLAY COMMINFO command (see "DISPLAY COMMINFO" on page 637).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

ENCODING
The encoding that the messages are transmitted in.

ASPUB
The encoding of the message is taken from the one that is supplied in the published message. This is the default value.

REVERSED

NORMAL

S390

TNS

encoding

GRPADDR
The group IP address or DNS name.

It is the administrator's responsibility to manage the group addresses. It is possible for all multicast clients to use the same group address for every topic; only the messages that match outstanding subscriptions on the client are delivered. Using the same group address can be inefficient because every client must examine and process every multicast packet in the network. It is more efficient to allocate different IP group addresses to different topics or sets of topics, but this requires careful management, especially if other non-MQ multicast applications are in use on the network. The default value is `239.0.0.0`.

MCHBINT
The heartbeat interval is measured in milliseconds, and specifies the frequency at which the transmitter notifies any receivers that there is no further data available. The value is in the range 0 to 999 999. The default value is `2000` milliseconds.

MCPROP
The multicast properties control how many of the MQMD properties and user properties flow with the message.

All
All user properties and all the fields of the MQMD are transported.

Reply
Only user properties, and MQMD fields that deal with replying to the messages, are transmitted. These properties are:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

User
Only the user properties are transmitted.

NONE
No user properties or MQMD fields are transmitted.

COMPAT
This value causes the transmission of the message to be done in a compatible mode to RMM. This allows some inter-operation with the current XMS applications and Broker RMM applications.

MONINT(*integer***)**
How frequently, in seconds, that monitoring information is updated. If events messages are enabled, this parameter also controls how frequently event messages are generated about the status of the Multicast handles created using this COMMINFO object.

A value of 0 means that there is no monitoring.

The default value is 60.

MSGHIST
This value is the amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs (negative acknowledgments).

The value is in the range 0 to 999 999 999. A value of `0` gives the least level of reliability. The default value is `100`.

NSUBHIST
The new subscriber history controls whether a subscriber joining a publication stream receives as much data as is currently available, or receives only publications made from the time of the subscription.

NONE
A value of `NONE` causes the transmitter to transmit only publication made from the time of the subscription. This is the default value.

ALL
A value of `ALL` causes the transmitter to retransmit as much history of the topic as is known. In some circumstances this can give a similar behavior to retained publications.

Note: Using the value of `ALL` might have a detrimental effect on performance if there is a large topic history because all the topic history is retransmitted.

PORT(*integer***)**
The port number to transmit on. The default port number is `1414`.

LIKE(*authinfo-name***)**
The name of a communication information object, with parameters that are used to model this definition.

If this field is not complete, and you do not complete the parameter fields related to the command, the values are taken from the default definition for an object of this type.

This default communication information object definition can be altered by the installation to the default values required.

REPLACE and NOREPLACE
Whether the existing definition is to be replaced with this one. This is optional. The default is NOREPLACE. Any object with a different disposition is not changed.

REPLACE
The definition replaces an existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE
The definition does not replace an existing definition of the same name.

DEFINE LISTENER:

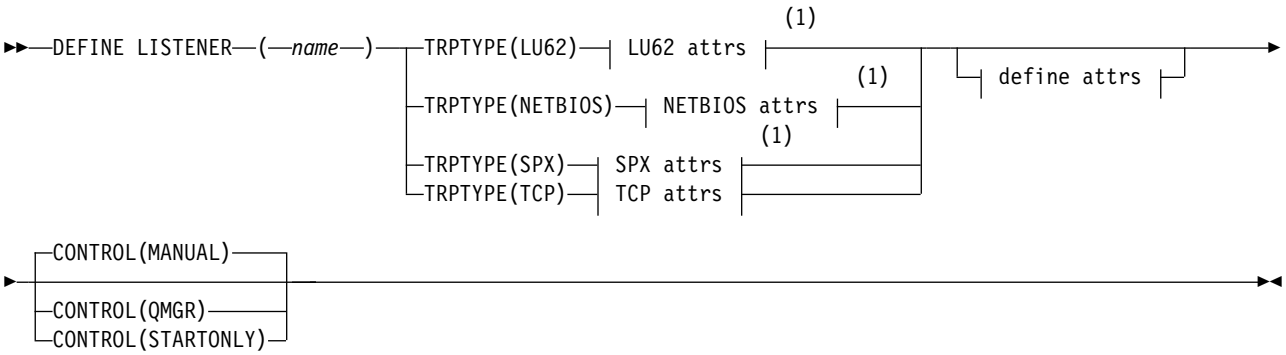
Use the MQSC command DEFINE LISTENER to define a new WebSphere MQ listener definition, and set its parameters.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for DEFINE LISTENER" on page 501

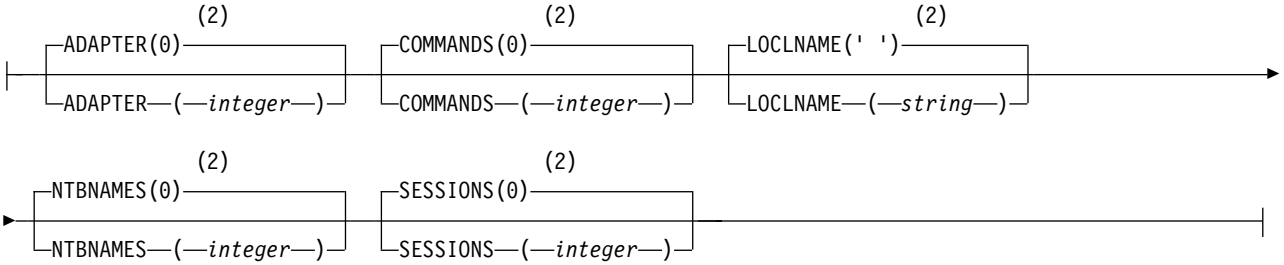
Synonym: DEF LSTR

DEFINE LISTENER

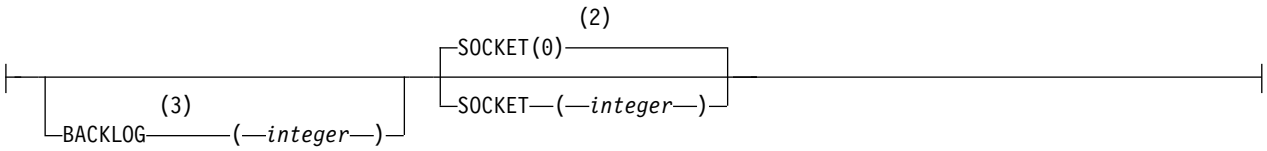


LU62 attrs:

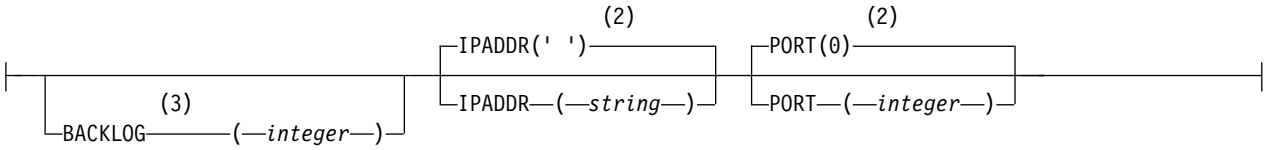
NETBIOS attrs:



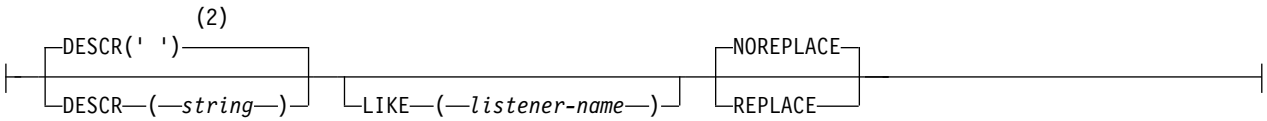
SPX attrs:



TCP attrs:



Define attrs:



Notes:

1 Valid only on Windows.

2 This is the default supplied with WebSphere MQ, but your installation might have changed it.

3 When the BACKLOG attribute is left unchanged or when it is explicitly set to zero, the attribute is stored as zero by default in the listener object created by the **DEFINE LISTENER** command. However, when the listener is started, the default backlog value takes effect. For information about the default value of the BACKLOG attribute, see Using the TCP listener backlog option.

Parameter descriptions for DEFINE LISTENER

(listener-name)

Name of the WebSphere MQ listener definition (see Rules for naming IBM WebSphere MQ objects). This is required.

The name must not be the same as any other listener definition currently defined on this queue manager (unless REPLACE is specified).

ADAPTER(*integer***)**
The adapter number on which NetBIOS listens. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

BACKLOG(*integer***)**
The number of concurrent connection requests that the listener supports.

COMMANDS(*integer***)**
The number of commands that the listener can use. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

CONTROL(*string***)**
Specifies how the listener is to be started and stopped.:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the listener when an operator issues the DISPLAY LISTENER command (see "DISPLAY LISTENER" on page 654).

It should contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

IPADDR(*string***)**
IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form. If you do not specify a value for this parameter, the listener listens on all configured IPv4 and IPv6 stacks.

LIKE(*listener-name***)**
The name of a listener, with parameters that are used to model this definition.

This parameter applies only to the DEFINE LISTENER command.

If this field is not filled in, and you do not complete the parameter fields related to the command, the values are taken from the default definition for listeners on this queue manager. This is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.LISTENER)
```

A default listener is provided but it can be altered by the installation of the default values required. See Rules for naming IBM WebSphere MQ objects .

LOCLNAME(*string***)**
The NetBIOS local name that the listener uses. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

NTBNAMES(*integer***)**
The number of names that the listener can use. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

PORT(*integer***)**
The port number for TCP/IP. This is valid only when TRPTYPE is TCP. It must not exceed 65535.

SESSIONS(*integer***)**
The number of sessions that the listener can use. This parameter is valid only on Windows when TRPTYPE is NETBIOS.

SOCKET(*integer***)**
The SPX socket on which to listen. This is valid only if TRPTYPE is SPX.

TPNAME(*string***)**
The LU 6.2 transaction program name (maximum length 64 characters). This parameter is valid only on Windows when TRPTYPE is LU62.

TRPTYPE(*string***)**
The transmission protocol to be used:

LU62
SNA LU 6.2. This is valid only on Windows.

NETBIOS
NetBIOS. This is valid only on Windows.

SPX
Sequenced packet exchange. This is valid only on Windows.

TCP
TCP/IP.

DEFINE NAMELIST:

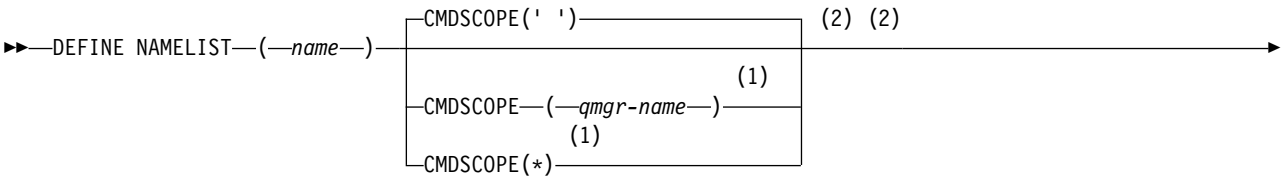
Use the MQSC command DEFINE NAMELIST to define a list of names. This is most commonly a list of cluster names or queue names.

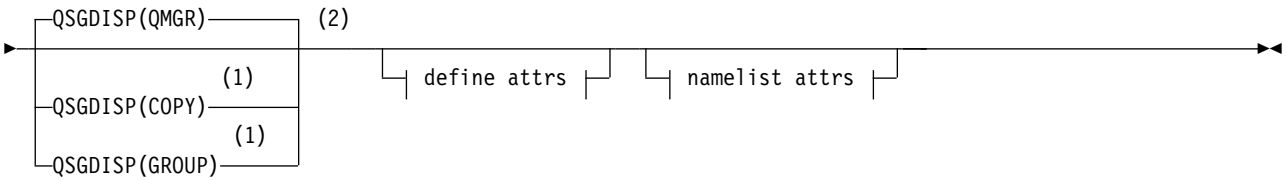
UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes" on page 504
- "Parameter descriptions for DEFINE NAMELIST" on page 504

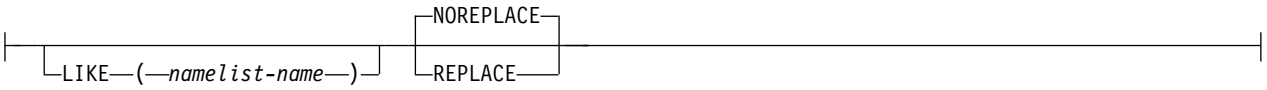
Synonym: DEF NL

DEFINE NAMELIST

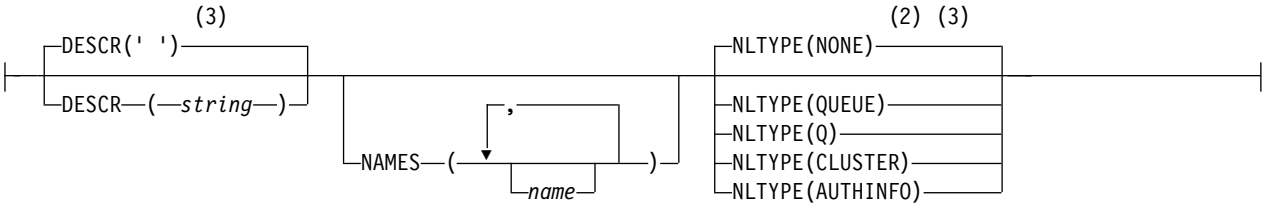




Define attrs:



Namelist attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 This is the default supplied with WebSphere MQ, but your installation might have changed it.

Usage notes

On UNIX systems, the command is valid only on AIX, HP-UX, Linux and Solaris.

Parameter descriptions for DEFINE NAMELIST

(name) Name of the list.

The name must not be the same as any other namelist name currently defined on this queue manager (unless REPLACE or ALTER is specified). See Rules for naming IBM WebSphere MQ objects.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

'' The command is executed on the queue manager on which it was entered.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active

queue manager in the queue-sharing group. The effect of specifying * is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the namelist when an operator issues the DISPLAY NAMELIST command (see "DISPLAY NAMELIST" on page 661).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LIKE(*namelist-name***)**
The name of a namelist, with parameters that are used to model this definition.

If this field is not completed and you do not complete the parameter fields related to the command, the values are taken from the default definition for namelists on this queue manager.

Not completing this parameter is equivalent to specifying:
`LIKE(SYSTEM.DEFAULT.NAMELIST)`

A default namelist definition is provided, but it can be altered by the installation to the default values required. See Rules for naming IBM WebSphere MQ objects.

On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified.

NAMES(*name, ...***)**
List of names.

The names can be of any type, but must conform to the rules for naming WebSphere MQ objects, with a maximum length of 48 characters.

An empty list is valid: specify NAMES(). The maximum number of names in the list is 256.

NLTYPE
Indicates the type of names in the namelist.

This parameter is valid only on z/OS.

`NONE`
The names are of no particular type.

`QUEUE or Q`
A namelist that holds a list of queue names.

`CLUSTER`
A namelist that is associated with clustering, containing a list of the cluster names.

`AUTHINFO`
This namelist is associated with SSL and contains a list of authentication information object names.

Namelists used for clustering must have NLTYPE(CLUSTER) or NLTYPE(NONE).

Namelists used for SSL must have NLTYPE(AUTHINFO).

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the 'LIKE' object.

GROUP	The object definition resides in the shared repository but only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero:
	`DEFINE NAMELIST(name)`
	`REPLACE QSGDISP(COPY)`
	
	The DEFINE for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE

Whether the existing definition (and on z/OS, with the same disposition) is to be replaced with this one. Any object with a different disposition is not changed.

REPLACE

The definition replaces any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE

The definition does not replace any existing definition of the same name.

DEFINE PROCESS:

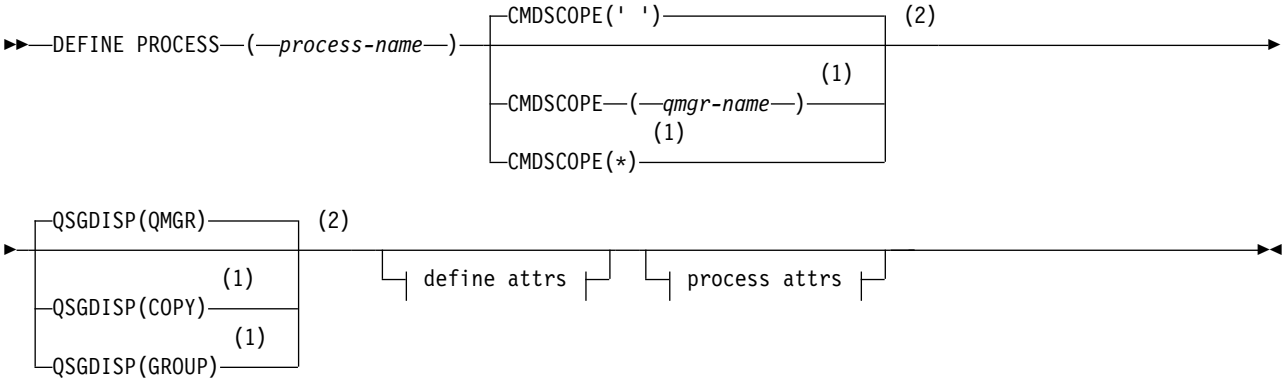
Use the MQSC command DEFINE PROCESS to define a new WebSphere MQ process definition, and set its parameters.

UNIX and Linux	Windows
✓	✓

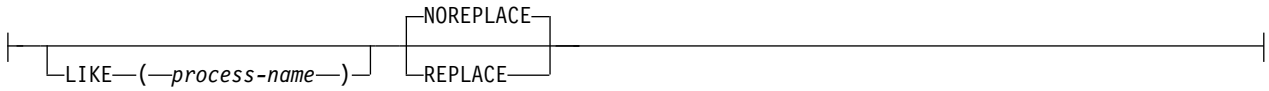
- Syntax diagram
- “Parameter descriptions for DEFINE PROCESS” on page 508

Synonym: DEF PRO

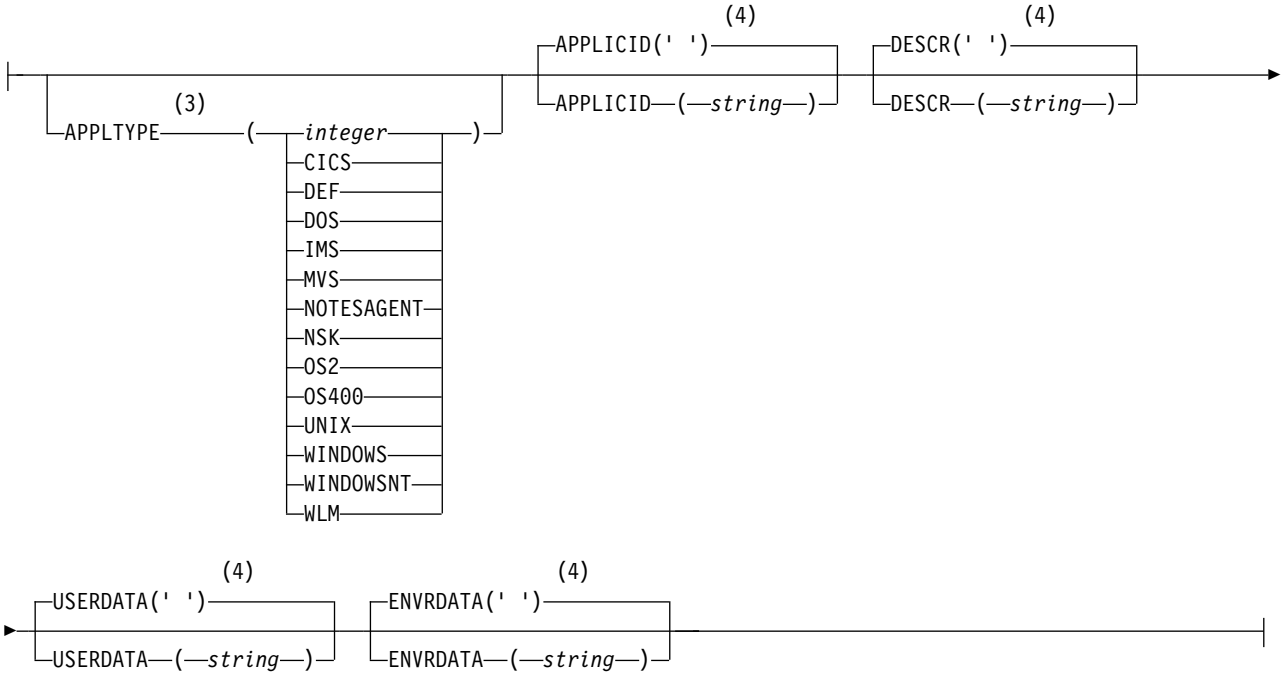
DEFINE PROCESS



Define attrs:



Process attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 The default depends on the platform, and can be changed by your installation.

4 This is the default supplied with WebSphere MQ, but your installation might have changed it.

Parameter descriptions for DEFINE PROCESS

(process-name)
Name of the WebSphere MQ process definition (see Rules for naming IBM WebSphere MQ objects). *process-name* is required.

The name must not be the same as any other process definition currently defined on this queue manager (unless REPLACE is specified).

APPLICID(*string***)**
The name of the application to be started. The name might typically be a fully qualified file name of an executable object. Qualifying the file name is particularly important if you have multiple IBM WebSphere MQ installations, to ensure the correct version of the application is run. The maximum length is 256 characters.

For a CICS application the name is a CICS transaction ID, and for an IMS application it is an IMS transaction ID.

On z/OS, for distributed queuing, it must be **CSQX START**.

APPLTYPE(*string***)**
The type of application to be started. Valid application types are:

integer
A system-defined application type in the range zero through 65 535 or a user-defined application type in the range 65 536 through 999 999 999.

For certain values in the system range, a parameter from the following list can be specified instead of a numeric value:

CICS Represents a CICS transaction.

DOS Represents a DOS application.

IMS Represents an IMS transaction.

MVS Represents a z/OS application (batch or TSO).

NOTESAGENT
Represents a Lotus Notes agent.

NSK Represents an HP Integrity NonStop Server application.

OS400 Represents an IBM i application.

UNIX Represents a UNIX application.

WINDOWS
Represents a Windows application.

WINDOWSNT
Represents a Windows NT, Windows 2000, or Windows XP application.

WLM Represents a z/OS workload manager application.

DEF Specifying DEF causes the default application type for the platform at which the command is interpreted to be stored in the process definition. This default cannot be changed by the installation. If the platform supports clients, the default is interpreted as the default application type of the server.

Only use application types (other than user-defined types) that are supported on the platform at which the command is executed:

- On z/OS, CICS, DOS, IMS, MVS, OS2, UNIX, WINDOWS, WINDOWSNT, WLM, and DEF are supported
- On IBM i, OS400, CICS, and DEF are supported
- On UNIX systems, UNIX, OS2, DOS, WINDOWS, CICS, and DEF are supported

- On Windows, WINDOWSNT, DOS, WINDOWS, OS2, UNIX, CICS, and DEF are supported

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

In a shared queue environment, you can provide a different queue manager name from the one you are using to enter the command. The command server must be enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

DESCR(*string***)**

Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY PROCESS command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

ENVRDATA(*string***)**

A character string that contains environment information pertaining to the application to be started. The maximum length is 128 characters.

The meaning of ENVRDATA is determined by the trigger-monitor application. The trigger monitor provided byIBM WebSphere MQ appends ENVRDATA to the parameter list passed to the started application. The parameter list consists of the MQTMC2 structure, followed by one blank, followed by ENVRDATA with trailing blanks removed.

Note:

1. On z/OS, ENVRDATA is not used by the trigger-monitor applications provided by IBM WebSphere MQ.
2. On z/OS, if APPLTYPE is WLM, the default values for the ServiceName and ServiceStep fields in the work information header (MQWIH) can be supplied in ENVRDATA. The format must be:

   ```
   SERVICENAME=servname,SERVICESTEP=stepname
   ```

   where:

   SERVICENAME=
   is the first 12 characters of ENVRDATA.

   servname
   is a 32-character service name. It can contain embedded blanks or any other data, and have trailing blanks. It is copied to the MQWIH as is.

   SERVICESTEP=
   is the next 13 characters of ENVRDATA.

stepname
is a 1 - 8 character service step name. It is copied as-is to the MQWIH, and padded to eight characters with blanks.

If the format is incorrect, the fields in the MQWIH are set to blanks.

3. On UNIX systems, ENVRDATA can be set to the ampersand character to make the started application run in the background.

LIKE(*process-name***)**
The name of an object of the same type, with parameters that are used to model this definition.

If this field is not provided, the values of fields you do not provide are taken from the default definition for this object.

Using `LIKE` is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.PROCESS)
```

A default definition for each object type is provided. You can alter the provided defaults to the default values required. See Rules for naming IBM WebSphere MQ objects.

On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified.

QSGDISP
This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command. It uses the QSGDISP(GROUP) object of the same name as the 'LIKE' object.

GROUP	The object definition resides in the shared repository. `GROUP` is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated.
	`DEFINE PROCESS(process-name)`
	`REPLACE QSGDISP(COPY)`
	
	The command is sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero. The DEFINE for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE
Whether the existing definition (and on z/OS, with the same disposition) is to be replaced with this one. REPLACE is optional. Any object with a different disposition is not changed.

REPLACE
The definition replaces any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE
The definition does not replace any existing definition of the same name.

USERDATA(*string***)**
A character string that contains user information pertaining to the application defined in the APPLICID that is to be started. The maximum length is 128 characters.

The meaning of USERDATA is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ simply passes USERDATA to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing USERDATA), followed by one blank, followed by ENVRDATA with trailing blanks removed.

For WebSphere MQ message channel agents, the format of this field is a channel name of up to 20 characters. See Managing objects for triggering for information about what APPLICID to provide to message channel agents.

For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

DEFINE queues:

Use the MQSC **DEFINE** command to define a local, model, or remote queue, or a queue alias, reply-to queue alias, or a queue-manager alias.

This section contains the following commands:
- "DEFINE QALIAS" on page 535
- "DEFINE QLOCAL" on page 536
- "DEFINE QMODEL" on page 539
- "DEFINE QREMOTE" on page 542

Define a reply-to queue or queue-manager alias with the "DEFINE QREMOTE" on page 542 command.

These commands are supported on the following platforms:

UNIX and Linux	Windows
✓	✓

Usage notes for DEFINE queues

1. For local queues
   a. You can define a local queue with QSGDISP(SHARED) even though another queue manager in the queue-sharing group already has a local version of the queue. However, when you try to access the locally defined queue, it fails with reason code MQRC_OBJECT_NOT_UNIQUE (2343). A local version of the queue with the same name can be of type QLOCAL, QREMOTE, or QALIAS and has the disposition, QSGDISP(QMGR).

      To resolve the conflict, you must delete one of the queues using the **DELETE** command. If the queue you want to delete contains messages, use the PURGE option or remove the messages first using the **MOVE** command.

      For example, to delete the QSGDISP(LOCAL) version, which contains messages, and copy those messages to the QSGDISP(SHARED) version, then issue the following commands:

```
MOVE QLOCAL(QUEUE.1) QSGDISP(PRIVATE) TOQLOCAL(QUEUE.1) TYPE(ADD)
DELETE QLOCAL(QUEUE.1) QSGDISP(QMGR)
```

2. For alias queues:
   a. `DEFINE QALIAS(aliasqueue) TARGET(otherqname) CLUSTER(c)` advertises the queue *otherqname* by the name *aliasqueue*.
   b. `DEFINE QALIAS(aliasqueue) TARGET(otherqname)` allows a queue advertised by the name *otherqname* to be used on this queue manager by the name *aliasqueue*.
   c. `TARGTYPE` and `TARGET` are not cluster attributes, that is, they are not shared in a cluster environment.
3. For remote queues:
   a. `DEFINE QREMOTE(rqueue) RNAME(otherq) RQMNAME(otherqm) CLUSTER(cl)` advertises this queue manager as a store and forward gateway to which messages for queue *rqueue* can be sent. It has no effect as a reply-to queue alias, except on the local queue manager.

      `DEFINE QREMOTE(otherqm) RNAME() RQMNAME(anotherqm) XMITQ(xq) CLUSTER` advertises this queue manager as a store and forward gateway to which messages for *anotherqm* can be sent.
   b. `RQMNAME` can itself be the name of a cluster queue manager within the cluster. You can map the advertised queue manager name to another name locally. The pattern is the same as with `QALIAS` definitions.
   c. It is possible for the values of `RQMNAME` and `QREMOTE` to be the same if `RQMNAME` is itself a cluster queue manager. If this definition is also advertised using a `CLUSTER` attribute, do not choose the local queue manager in the cluster workload exit. If you do so, a cyclic definition results.
   d. Remote queues do not have to be defined locally. The advantage of doing so is that applications can refer to the queue by a simple, locally defined name. If you do then the queue name is qualified by the name of the queue manager on which the queue resides. Using a local definition means that applications do not need to be aware of the real location of the queue.
   e. A remote queue definition can also be used as a mechanism for holding a queue manager alias definition, or a reply-to queue alias definition. The name of the definition in these cases is:
      - The queue manager name being used as the alias for another queue manager name (queue manager alias), or
      - The queue name being used as the alias for the reply-to queue (reply-to queue alias).

Parameter descriptions for DEFINE QUEUE and ALTER QUEUE

Table 79 shows the parameters that are relevant for each type of queue. There is a description of each parameter after the table.

Table 79. DEFINE and ALTER QUEUE parameters.

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

Parameter	Local queue	Model queue	Alias queue	Remote queue
ACCTQ	✓	✓		
BOQNAME	✓	✓		
BOTHRESH	✓	✓		
CFSTRUCT	✓	✓		
CLCHNAME	✓	✓		
CLUSNL	✓		✓	✓

Table 79. DEFINE and ALTER QUEUE parameters (continued).

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

Parameter	Local queue	Model queue	Alias queue	Remote queue
CLUSTER	✓		✓	✓
CLWLPRTY	✓		✓	✓
CLWLRANK	✓		✓	✓
CLWLUSEQ	✓			
CMDSCOPE	✓	✓	✓	✓
CUSTOM	✓	✓	✓	✓
DEFBIND	✓		✓	✓
DEFPRESP	✓	✓	✓	✓
DEFPRTY	✓	✓	✓	✓
DEFPSIST	✓	✓	✓	✓
DEFREADA	✓	✓	✓	
DEFSOPT	✓	✓		
DEFTYPE		✓		
DESCR	✓	✓	✓	✓
DISTL	✓	✓		
FORCE	✓		✓	✓
GET	✓	✓	✓	
HARDENBO or NOHARDENBO	✓	✓		
INDXTYPE	✓	✓		
INITQ	✓	✓		
LIKE	✓	✓	✓	✓
MAXDEPTH	✓	✓		
MAXMSGL	✓	✓		

Table 79. DEFINE and ALTER QUEUE parameters (continued).

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

Parameter	Local queue	Model queue	Alias queue	Remote queue
MONQ	✓	✓		
MSGDLVSQ	✓	✓		
NOREPLACE	✓	✓	✓	✓
NPMCLASS	✓	✓		
PROCESS	✓	✓		
PROPCTL	✓	✓	✓	
PUT	✓	✓	✓	✓
queue-name	✓	✓	✓	✓
QDEPTHHI	✓	✓		
QDEPTHLO	✓	✓		
QDPHIEV	✓	✓		
QDPLOEV	✓	✓		
QDPMAXEV	✓	✓		
QSGDISP	✓	✓	✓	✓
QSVCIEV	✓	✓		
QSVCINT	✓	✓		
REPLACE	✓	✓	✓	✓
RETINTVL	✓	✓		
RNAME				✓
RQMNAME				✓
SCOPE	✓		✓	✓
SHARE or NOSHARE	✓	✓		
STATQ	✓	✓		

Table 79. DEFINE and ALTER QUEUE parameters (continued).

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

Parameter	Local queue	Model queue	Alias queue	Remote queue
STGCLASS	✓	✓		
TARGET			✓	
TARGQ			✓	
TARGTYPE			✓	
TRIGDATA	✓	✓		
TRIGDPTH	✓	✓		
TRIGGER or NOTRIGGER	✓	✓		
TRIGMPRI	✓	✓		
TRIGTYPE	✓	✓		
USAGE	✓	✓		
XMITQ				✓

queue-name
Local name of the queue, except the remote queue where it is the local definition of the remote queue.

The name must not be the same as any other queue name of any queue type currently defined on this queue manager, unless you specify **REPLACE** or **ALTER**. See Rules for naming IBM WebSphere MQ objects.

ACCTQ
Specifies whether accounting data collection is to be enabled for the queue. On z/OS, the data collected is class 3 accounting data (thread-level and queue-level accounting). In order for accounting data to be collected for this queue, accounting data for this connection must also be enabled. Turn on accounting data collection by setting either the **ACCTQ** queue manager attribute, or the options field in the MQCNO structure on the MQCONNX call.

QMGR The collection of accounting data is based on the setting of the **ACCTQ** parameter on the queue manager definition.

ON Accounting data collection is enabled for the queue unless the **ACCTQ** queue manager parameter has a value of NONE. On z/OS systems, you must switch on class 3 accounting using the **START TRACE** command.

OFF Accounting data collection is disabled for the queue.

BOQNAME(*queue-name***)**
The excessive backout requeue name.

This parameter is supported only on local and model queues.

Use this parameter to set or change the back out queue name attribute of a local or model queue. Apart from allowing its value to be queried, the queue manager does nothing based on the value of this attribute. IBM WebSphere MQ classes for JMS transfers a message that is backed out the maximum number of times to this queue. The maximum is specified by the **BOTHRESH** attribute.

BOTHRESH(*integer***)**
The backout threshold.

This parameter is supported only on local and model queues.

Use this parameter to set or change the value of the back out threshold attribute of a local or model queue. Apart from allowing its value to be queried, the queue manager does nothing based on the value of this attribute. IBM WebSphere MQ classes for JMS use the attribute to determine how many times back a message out. When the value is exceeded the message is transferred to the queue named by the **BOQNAME** attribute.

Specify a value in the range 0 - 999,999,999.

CFSTRUCT(*structure-name***)**
Specifies the name of the coupling facility structure where you want messages stored when you use shared queues.

This parameter is supported only on z/OS for local and model queues.

The name:

- Cannot have more than 12 characters
- Must start with an uppercase letter (A - Z)
- Can include only the characters A - Z and 0 - 9

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always four characters, padded with @ symbols if necessary. For example, if you use a queue-sharing group named NY03 and you supply the name PRODUCT7, the resultant coupling facility structure name is NY03PRODUCT7. The administrative structure for the queue-sharing group (in this case NY03CSQ_ADMIN) cannot be used for storing messages.

For ALTER QLOCAL, ALTER QMODEL, DEFINE QLOCAL with **REPLACE**, and DEFINE QMODEL with **REPLACE** the following rules apply:

- On a local queue with **QSGDISP**(SHARED), **CFSTRUCT** cannot change.

  If you change either the **CFSTRUCT** or **QSGDISP** value you must delete and redefine the queue. To preserve any of the messages on the queue you must offload the messages before you delete the queue. Reload the messages after you redefine the queue, or move the messages to another queue.
- On a model queue with **DEFTYPE**(SHAREDYN), **CFSTRUCT** cannot be blank.
- On a local queue with a **QSGDISP** other than SHARED, or a model queue with a **DEFTYPE** other than SHAREDYN, the value of **CFSTRUCT** does not matter.

For DEFINE QLOCAL with **NOREPLACE** and DEFINE QMODEL with **NOREPLACE**, the coupling facility structure:

- On a local queue with **QSGDISP**(SHARED) or a model queue with a **DEFTYPE**(SHAREDYN), **CFSTRUCT** cannot be blank.
- On a local queue with a **QSGDISP** other than SHARED, or a model queue with a **DEFTYPE** other than SHAREDYN, the value of **CFSTRUCT** does not matter.

Note: Before you can use the queue, the structure must be defined in the coupling facility Resource Management (CFRM) policy data set.

CLCHNAME(*channel name***)**

This parameter is supported only on transmission queues.

CLCHNAME is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue. CLCHNAME is not supported on z/OS.

You can also set the transmission queue attribute CLCHNAME attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the CLCHNAME attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either SYSTEM.CLUSTER.TRANSMIT.*ChannelName* or SYSTEM.CLUSTER.TRANSMIT.QUEUE, depending on the value of the queue manager DEFCLXQ attribute.

By specifying asterisks, “"*"”, in CLCHNAME, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. CLCHNAME is limited to a length of 48 characters, MQ_OBJECT_NAME_LENGTH. A channel name is limited to 20 characters: MQ_CHANNEL_NAME_LENGTH.

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. The default configuration can be changed by modified by changing the queue manager attribute, DEFCLXQ. The default value of the attribute is SCTQ. You can change the value to CHANNEL. If you set the DEFCLXQ attribute to CHANNEL, each cluster-sender channel defaults to using a specific cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

CLUSNL(*namelist name***)**
The name of the namelist that specifies a list of clusters to which the queue belongs.

This parameter is supported only on alias, local, and remote queues.

Changes to this parameter do not affect instances of the queue that are already open.

Only one of the resultant values of **CLUSNL** or **CLUSTER** can be nonblank; you cannot specify a value for both.

On local queues, this parameter cannot be set for transmission, SYSTEM.CHANNEL.*xx*, SYSTEM.CLUSTER.*xx*, or SYSTEM.COMMAND.*xx* queues, and on z/OS only, for SYSTEM.QSG.*xx* queues.

This parameter is valid only on AIX, HP-UX, Linux, Solaris, Windows, and z/OS.

CLUSTER(*cluster name***)**
The name of the cluster to which the queue belongs.

This parameter is supported only on alias, local, and remote queues.

The maximum length is 48 characters conforming to the rules for naming IBM WebSphere MQ objects. Changes to this parameter do not affect instances of the queue that are already open.

Only one of the resultant values of **CLUSNL** or **CLUSTER** can be nonblank; you cannot specify a value for both.

On local queues, this parameter cannot be set for transmission, SYSTEM.CHANNEL.*xx*, SYSTEM.CLUSTER.*xx*, or SYSTEM.COMMAND.*xx* queues, and on z/OS only, for SYSTEM.QSG.*xx* queues.

This parameter is valid only on AIX, HP-UX, Linux, Solaris, Windows, and z/OS.

CLWLPRTY(*integer***)**
Specifies the priority of the queue for the purposes of cluster workload distribution. This parameter is valid only for local, remote, and alias queues. The value must be in the range zero through 9 where zero is the lowest priority and 9 is the highest. For more information about this attribute, see CLWLPRTY queue attribute.

CLWLRANK(*integer***)**
Specifies the rank of the queue for the purposes of cluster workload distribution. This parameter

is valid only for local, remote, and alias queues. The value must be in the range zero through 9 where zero is the lowest rank and 9 is the highest. For more information about this attribute, see CLWLRANK queue attribute.

CLWLUSEQ
Specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance. The parameter has no effect when the MQPUT originates from a cluster channel. This parameter is valid only for local queues.

QMGR The behavior is as specified by the **CLWLUSEQ** parameter of the queue manager definition.

ANY The queue manager is to treat the local queue as another instance of the cluster queue for the purposes of workload distribution.

LOCAL The local queue is the only target of the MQPUT operation.

CMDSCOPE
This parameter applies to z/OS only. It specifies where the command is run when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if **QSGDISP** is set to GROUP or SHARED.

'' The command is run on the queue manager on which it was entered.

QmgrName
The command is run on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered. You can specify another name, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

CUSTOM(*string***)**
The custom attribute for new features.

This attribute is reserved for the configuration of new features before separate attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

This description is updated when features using this attribute are introduced. At the moment, there are no values for **CUSTOM**.

DEFBIND
Specifies the binding to be used when the application specifies MQOO_BIND_AS_Q_DEF on the MQOPEN call, and the queue is a cluster queue.

OPEN The queue handle is bound to a specific instance of the cluster queue when the queue is opened.

NOTFIXED
The queue handle is not bound to any instance of the cluster queue. The queue manager selects a specific queue instance when the message is put using MQPUT. It changes that selection later, if the need arises.

GROUP Allows an application to request that a group of messages is allocated to the same destination instance.

Multiple queues with the same name can be advertised in a queue manager cluster. An application can send all messages to a single instance, MQOO_BIND_ON_OPEN. It can allow a workload management algorithm to select the most suitable destination on a per message basis,

MQOO_BIND_NOT_FIXED. It can allow an application to request that a "group" of messages be all allocated to the same destination instance. The workload balancing reselects a destination between groups of messages, without requiring an MQCLOSE and MQOPEN of the queue.

The MQPUT1 call always behaves as if NOTFIXED is specified.

This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

DEFPRESP
Specifies the behavior to be used by applications when the put response type, within the MQPMO options, is set to MQPMO_RESPONSE_AS_Q_DEF.

SYNC Put operations to the queue specifying MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_SYNC_RESPONSE is specified instead.

ASYNC Put operations to the queue specifying MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_ASYNC_RESPONSE is specified instead; see MQPMO options (MQLONG).

DEFPRTY(*integer*)
The default priority of messages put on the queue. The value must be in the range 0 - 9. Zero is the lowest priority, through to the **MAXPRTY** queue manager parameter. The default value of **MAXPRTY** is 9.

DEFPSIST
Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_Q_DEF option.

NO Messages on this queue are lost across a restart of the queue manager.

YES Messages on this queue survive a restart of the queue manager.

On z/OS, N and Y are accepted as synonyms of NO and YES.

DEFREADA
Specifies the default read ahead behavior for non-persistent messages delivered to the client. Enabling read ahead can improve the performance of client applications consuming non-persistent messages.

NO Non-persistent messages are not read ahead unless the client application is configured to request read ahead.

YES Non-persistent messages are sent to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not delete all the messages it is sent.

DISABLED
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

DEFSOPT
The default share option for applications opening this queue for input:

EXCL The open request is for exclusive input from the queue

SHARED The open request is for shared input from the queue

DEFTYPE
Queue definition type.

This parameter is supported only on model queues.

PERMDYN
A permanent dynamic queue is created when an application issues an MQOPEN MQI call with the name of this model queue specified in the object descriptor (MQOD).

On z/OS, the dynamic queue has a disposition of QMGR.

SHAREDYN
This option is available on z/OS only.

A permanent dynamic queue is created when an application issues an MQOPEN API call with the name of this model queue specified in the object descriptor (MQOD).

The dynamic queue has a disposition of SHARED.

TEMPDYN
A temporary dynamic queue is created when an application issues an MQOPEN API call with the name of this model queue specified in the object descriptor (MQOD).

On z/OS, the dynamic queue has a disposition of QMGR.

Do not specify this value for a model queue definition with a **DEFPSIST** parameter of YES.

If you specify this option, do not specify **INDXTYPE**(MSGTOKEN).

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY QUEUE command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters that are in the coded character set identifier (CCSID) of this queue manager. If you do not do so and if the information is sent to another queue manager, they might be translated incorrectly.

DISTL
DISTL sets whether distribution lists are supported by the partner queue manager.

YES Distribution lists are supported by the partner queue manager.

NO Distribution lists are not supported by the partner queue manager.

Note: You do not normally change this parameter, because it is set by the MCA. However you can set this parameter when defining a transmission queue if the distribution list capability of the destination queue manager is known.

This parameter is valid only on AIX, HP-UX, Linux, Solaris, and Windows.

FORCE
This parameter applies only to the ALTER command on alias, local and remote queues.

Specify this parameter to force completion of the command in the following circumstances.

For an alias queue, if both of the following are true:

- The **TARGQ** parameter is specified
- An application has this alias queue open

For a local queue, if both of the following are true:

- The **NOSHARE** parameter is specified
- More than one application has the queue open for input

FORCE is also needed if both of the following are true:

- The **USAGE** parameter is changed
- Either one or more messages are on the queue, or one or more applications have the queue open

Do not change the **USAGE** parameter while there are messages on the queue; the format of messages changes when they are put on a transmission queue.

For a remote queue, if both of the following are true:

- The **XMITQ** parameter is changed
- One or more applications has this queue open as a remote queue

FORCE is also needed if both of the following are true:

- Any of the **RNAME**, **RQMNAME**, or **XMITQ** parameters are changed
- One or more applications has a queue open that resolved through this definition as a queue manager alias

Note: **FORCE** is not required if this definition is in use as a reply-to queue alias only.

If **FORCE** is not specified in the circumstances described, the command is unsuccessful.

GET Specifies whether applications are to be permitted to get messages from this queue:

ENABLED
Messages can be retrieved from the queue, by suitably authorized applications.

DISABLED
Applications cannot retrieve messages from the queue.

This parameter can also be changed using the MQSET API call.

HARDENBO&NOHARDENBO
Specifies whether hardening is used to ensure that the count of the number of times that a message is backed out is accurate.

This parameter is supported only on local and model queues.

HARDENBO
The count is hardened.

NOHARDENBO
The count is not hardened.

Note: This parameter affects only IBM WebSphere MQ for z/OS. It can be set on other platforms but is ineffective.

INDXTYPE
The type of index maintained by the queue manager to expedite MQGET operations on the queue. For shared queues, the type of index determines the type of MQGET operations that can be used.

This parameter is supported only on local and model queues.

Messages can be retrieved using a selection criterion only if an appropriate index type is maintained, as the following table shows:

Index type required for different retrieval selection criteria

Retrieval selection criterion	**Index type required**	
	Shared queue	Other queue
None (sequential retrieval)	Any	Any
Message identifier	MSGID or NONE	Any
Correlation identifier	CORRELID	Any
Message and correlation identifiers	MSGID or CORRELID	Any
Group identifier	GROUPID	Any
Grouping	GROUPID	GROUPID

Index type required for different retrieval selection criteria

Retrieval selection criterion	Index type required	
Message token	Not allowed	MSGTOKEN

where the value of **INDXTYPE** parameter has the following values:

NONE No index is maintained. Use NONE when messages are typically retrieved sequentially or use both the message identifier and the correlation identifier as a selection criterion on the MQGET call.

MSGID An index of message identifiers is maintained. Use MSGID when messages are typically retrieved using the message identifier as a selection criterion on the MQGET call with the correlation identifier set to NULL.

CORRELID
An index of correlation identifiers is maintained. Use CORRELID when messages are typically retrieved using the correlation identifier as a selection criterion on the MQGET call with the message identifier set to NULL.

GROUPID
An index of group identifiers is maintained. Use GROUPID when messages are retrieved using message grouping selection criteria.

Note:

1. You cannot set **INDXTYPE** to GROUPID if the queue is a transmission queue.
2. The queue must use a CF structure at CFLEVEL(3), to specify a shared queue with **INDXTYPE**(GROUPID).

MSGTOKEN
An index of message tokens is maintained. Use MSGTOKEN when the queue is a WLM-managed queue that you are using with the Workload Manager functions of z/OS.

Note: You cannot set **INDXTYPE** to MSGTOKEN if:

- The queue is a model queue with a definition type of SHAREDYN
- The queue is a temporary dynamic queue
- The queue is a transmission queue
- You specify **QSGDISP**(SHARED)

For queues that are not shared and do not use grouping or message tokens, the index type does not restrict the type of retrieval selection. However, the index is used to expedite **GET** operations on the queue, so choose the type that corresponds to the most common retrieval selection.

If you are altering or replacing an existing local queue, you can change the **INDXTYPE** parameter only in the cases indicated in the following table:

Index type change permitted depending upon queue sharing and presence of messages in the queue.

Queue type		NON-SHARED			SHARED	
Queue state		Uncommitted activity	No uncommitted activity, messages present	No uncommitted activity, and empty	Open or messages present	Not open, and empty
Change INDXTYPE from:	To:	Change allowed?				
NONE	MSGID	No	Yes	Yes	No	Yes

Index type change permitted depending upon queue sharing and presence of messages in the queue.

Queue type		NON-SHARED			SHARED	
NONE	CORRELID	No	Yes	Yes	No	Yes
NONE	MSGTOKEN	No	No	Yes	-	-
NONE	GROUPID	No	No	Yes	No	Yes
MSGID	NONE	No	Yes	Yes	No	Yes
MSGID	CORRELID	No	Yes	Yes	No	Yes
MSGID	MSGTOKEN	No	No	Yes	-	-
MSGID	GROUPID	No	No	Yes	No	Yes
CORRELID	NONE	No	Yes	Yes	No	Yes
CORRELID	MSGID	No	Yes	Yes	No	Yes
CORRELID	MSGTOKEN	No	No	Yes	-	-
CORRELID	GROUPID	No	No	Yes	No	Yes
MSGTOKEN	NONE	No	Yes	Yes	-	-
MSGTOKEN	MSGID	No	Yes	Yes	-	-
MSGTOKEN	CORRELID	No	Yes	Yes	-	-
MSGTOKEN	GROUPID	No	No	Yes	-	-
GROUPID	NONE	No	No	Yes	No	Yes
GROUPID	MSGID	No	No	Yes	No	Yes
GROUPID	CORRELID	No	No	Yes	No	Yes
GROUPID	MSGTOKEN	No	No	Yes	-	-

This parameter is supported only on z/OS. On other platforms, all queues are automatically indexed.

INITQ(*string***)**
The local name of the initiation queue on this queue manager, to which trigger messages relating to this queue are written; see Rules for naming IBM WebSphere MQ objects .

This parameter is supported only on local and model queues.

LIKE(*qtype-name***)**
The name of a queue, with parameters that are used to model this definition.

If this field is not completed, the values of undefined parameter fields are taken from one of the following definitions. The choice depends on the queue type:

Queue type	Definition
Alias queue	SYSTEM.DEFAULT.ALIAS.QUEUE
Local queue	SYSTEM.DEFAULT.LOCAL.QUEUE
Model queue	SYSTEM.DEFAULT.MODEL.QUEUE
Remote queue	SYSTEM.DEFAULT.REMOTE.QUEUE

For example, not completing this parameter is equivalent to defining the following value of LIKE for an alias queue:

```
LIKE(SYSTEM.DEFAULT.ALIAS.QUEUE)
```

If you require different default definitions for all queues, alter the default queue definitions instead of using the **LIKE** parameter.

On z/OS, the queue manager searches for an object with the name and queue type you specify with a disposition of QMGR, COPY, or SHARED. The disposition of the **LIKE** object is not copied to the object you are defining.

Note:

1. **QSGDISP** (GROUP) objects are not searched.
2. **LIKE** is ignored if **QSGDISP**(COPY) is specified.

MAXDEPTH(*integer***)**
The maximum number of messages allowed on the queue.

This parameter is supported only on local and model queues.

On AIX, HP-UX, Linux, Solaris, Windows, and z/OS, specify a value in the range zero through 999999999.

This parameter is valid only on AIX, HP-UX, Linux, Solaris, Windows, and z/OS.

On any other IBM WebSphere MQ platform, specify a value in the range zero through 640000.

Other factors can still cause the queue to be treated as full, for example, if there is no further hard disk space available.

If this value is reduced, any messages that are already on the queue that exceed the new maximum remain intact.

MAXMSGL(*integer***)**
The maximum length (in bytes) of messages on this queue.

This parameter is supported only on local and model queues.

On AIX, HP-UX, Linux, Solaris, and Windows, specify a value in the range zero to the maximum message length for the queue manager. See the **MAXMSGL** parameter of the ALTER QMGR command, ALTER QMGR MAXMSGL.

On z/OS, specify a value in the range zero through 100 MB (104 857 600 bytes).

Message length includes the length of user data and the length of headers. For messages put on the transmission queue, there are additional transmission headers. Allow an additional 4000 bytes for all the message headers.

If this value is reduced, any messages that are already on the queue with length that exceeds the new maximum are not affected.

Applications can use this parameter to determine the size of buffer for retrieving messages from the queue. Therefore, the value can be reduced only if it is known that this reduction does not cause an application to operate incorrectly.

Note that by adding the digital signature and key to the message, WebSphere MQ Advanced Message Security increases the length of the message.

MONQ
Controls the collection of online monitoring data for queues.

This parameter is supported only on local and model queues.

QMGR Collect monitoring data according to the setting of the queue manager parameter **MONQ**.

OFF Online monitoring data collection is turned off for this queue.

LOW If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

MEDIUM If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

HIGH If the value of the **MONQ** parameter of the queue manager is not NONE, online monitoring data collection is turned on for this queue.

There is no distinction between the values LOW, MEDIUM, and HIGH. These values all turn data collection on, but do not affect the rate of collection.

When this parameter is used in an ALTER queue command, the change is effective only when the queue is next opened.

MSGDLVSQ
Message delivery sequence.

This parameter is supported only on local and model queues.

PRIORITY
Messages are delivered (in response to MQGET API calls) in first-in-first-out (FIFO) order within priority.

FIFO Messages are delivered (in response to MQGET API calls) in FIFO order. Priority is ignored for messages on this queue.

The message delivery sequence parameter can be changed from PRIORITY to FIFO while there are messages on the queue. The order of the messages already on the queue is not changed. Messages added to the queue later take the default priority of the queue, and so might be processed before some of the existing messages.

If the message delivery sequence is changed from FIFO to PRIORITY, the messages put on the queue while the queue was set to FIFO take the default priority.

Note: If **INDXTYPE**(GROUPID) is specified with **MSGDLVSQ**(PRIORITY), the priority in which groups are retrieved is based on the priority of the first message within each group. The priorities 0 and 1 are used by the queue manager to optimize the retrieval of messages in logical order. The first message in each group must not use these priorities. If it does, the message is stored as if it was priority two.

NPMCLASS
The level of reliability to be assigned to non-persistent messages that are put to the queue:

NORMAL Non-persistent messages are lost after a failure, or queue manager shutdown. These messages are discarded on a queue manager restart.

HIGH The queue manager attempts to retain non-persistent messages on this queue over a queue manager restart or switch over.

You cannot set this parameter on z/OS.

PROCESS(*string***)**
The local name of the IBM WebSphere MQ process.

This parameter is supported only on local and model queues.

This parameter is the name of a process instance that identifies the application started by the queue manager when a trigger event occurs; see Rules for naming IBM WebSphere MQ objects .

The process definition is not checked when the local queue is defined, but it must be available for a trigger event to occur.

If the queue is a transmission queue, the process definition contains the name of the channel to be started. This parameter is optional for transmission queues on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS. If you do not specify it, the channel name is taken from the value specified for the **TRIGDATA** parameter.

PROPCTL
Property control attribute. The attribute is optional. It is applicable to local, alias, and model

queues. For information about how an administrator can use **PROPCTL** to control applications that access message properties, see Using the **PROPCTL** queue property to control message properties. Using the **PROPCTL** channel property to control message properties explains how to use **PROPCTL** to control the transfer of properties to queue managers running at Version 6.0 or earlier.

PROPCTL options are as follows. The options do not affect message properties in the MQMD or MQMD extension.

ALL

Set ALL so that an application can read all the properties of the message either in MQRFH2 headers, or as properties of the message handle.

The ALL option enables applications that cannot be changed to access all the message properties from MQRFH2 headers. Applications that can be changed, can access all the properties of the message as properties of the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

COMPAT

Set COMPAT so that unmodified applications that expect JMS-related properties to be in an MQRFH2 header in the message data continue to work as before. Applications that can be changed, can access all the properties of the message as properties of the message handle.

If the message contains a property with a prefix of mcd., jms., usr., or mqext., all message properties are delivered to the application. If no message handle is supplied, properties are returned in an MQRFH2 header. If a message handle is supplied, all properties are returned in the message handle.

If the message does not contain a property with one of those prefixes, and the application does not provide a message handle, no message properties are returned to the application. If a message handle is supplied, all properties are returned in the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

FORCE

Force all applications to read message properties from MQRFH2 headers.

Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible using the message handle.

In some cases, the format of data in MQRFH2 headers in the received message might be different to the format in the message when it was sent.

NONE

If a message handle is supplied, all the properties are returned in the message handle.

All message properties are removed from the message body before it is delivered to the application.

V6COMPAT

Set V6COMPAT so that applications that expect to receive the same MQRFH2 created by a sending application, can receive it as it was sent. The data in the MQRFH2 header is subject to character set conversion and numeric encoding changes. If the application sets properties using MQSETMP, the properties are not added to the MQRFH2 header created by

the application. The properties are accessible only by using the MQINQMP call. The properties are transmitted in an extra MQRFH2 that is visible to channel exits, but not to MQI programs. If properties are inserted in the MQRFH2 header by the sending application, they are only accessible to the receiving application in the MQRFH2 header. You cannot query properties set this way by calling MQINQMP. This behavior of properties and application created MQRFH2 headers occurs only when V6COMPAT is set.

The receiving application can override the setting of V6COMPAT, by setting an MQGMO_PROPERTIES option, such as MQGMO_PROPERTIES_IN_HANDLE. The default setting of MQGMO_PROPERTIES is MQGMO_PROPERTIES_AS_Q_DEF, which leaves the property setting as defined by the **PROPCTL** setting on the resolved receiving queue.

Note: If the **PSPROP** subscription attribute is set to RFH2, the queue manager might add publish/subscribe properties to the psc folder in the application-created MQRFH2 header. Otherwise, the queue manager does not modify the application-created MQRFH2 header.

Special rules apply to setting V6COMPAT:

1. You must set V6COMPAT on both of the queues accessed by MQPUT and MQGET.
   - You might find the effect of V6COMPAT does not require setting V6COMPAT on the queue that MQPUT writes to. The reason is that in many cases MQPUT does not reorganize the contents of an MQRFH2. Setting V6COMPAT has no apparent effect.
   - V6COMPAT appears to take effect only when it is set on the queue that is accessed by the application receiving the message.

   Despite these appearances, it is important you set V6COMPAT for both the sender and receiver of a message. In some circumstances, V6COMPAT works only if it is set at both ends of the transfer.
2. If you set V6COMPAT on either an alias queue or a local queue, the result is the same.

   For example, an alias queue, QA1, has a target queue Q1. An application opens QA1. Whichever of the pairs of definitions in Figure 5 on page 401 are set, the result is the same. A message is placed on Q1, with the MQRFH2 created by the application preserved exactly as it was when it was passed to the queue manager.

```
DEFINE QLOCAL(Q1)  PROPCTL(V6COMPAT)
DEFINE QALIAS(QA1) TARGET(Q1)
```

```
DEFINE QLOCAL(Q1)
DEFINE QALIAS(QA1) TARGET(Q1) PROPCTL(V6COMPAT)
```

Figure 6. Equivalent definitions of V6COMPAT

3. You can set V6COMPAT on the transmission queue, or a queue that resolves to a transmission queue. The result is to transmit any MQRFH2 in a message exactly as it was created by an application. You cannot set V6COMPAT on a QREMOTE definition.

   No other **PROPCTL** queue options behave this way. To control the way message properties are transmitted to a queue manager running IBM WebSphere MQ Version 6.0 or earlier, set the **PROPCTL** channel attribute; see Using the **PROPCTL** channel property to control message properties.
4. For publish/subscribe, V6COMPAT must be set on a queue that resolves to the destination for a publication.
   - For unmanaged publish/subscribe, set V6COMPAT on a queue that is in the name resolution path for the queue passed to MQSUB. If a subscription is created administratively, set V6COMPAT on a queue that is in the name resolution path for the destination set for the subscription.

- For managed publish/subscribe, set V6COMPAT on the model managed durable and managed non-durable queues for subscription topics. The default model managed queues are SYSTEM.MANAGED.DURABLE and SYSTEM.MANAGED.NDURABLE. By using different model queues for different topics, some publications are received with their original MQRFH2, and others with message property control set by other values of **PROPCTL**.
- For queued publish/subscribe, you must identify the queues used by the publishing and subscribing applications. Set V6COMPAT on those queues, as if the publisher and subscriber are using point to point messaging.

The effect of setting V6COMPAT on a message sent to another queue manager is as follows:

To a Version 7.1 queue manager

If a message contains internally set message properties, or message properties set by MQSETMP, the local queue manager adds an MQRFH2. The additional MQRFH2 is placed before any application created MQRFH2 headers. The local queue manager passes the modified message to the channel.

The new MQRFH2 header is flagged MQRFH_INTERNAL (X'8000000') in the MQRFH2 Flags field; see Flags (MQLONG) .

The channel message, and send and receive exits, are passed the entire message including the additional MQRFH2.

The action of the remote channel depends on whether V6COMPAT is set for the target queue. If it is set, then the internally set properties in the initial MQRFH2 are available to an application in the message handle. The application created MQRFH2 is received unchanged, except for character conversion and numeric encoding transformations.

To a Version 7.0.1 queue manager
Internally set properties are discarded. The MQRFH2 header is transferred unmodified.

To a Version 6.0 or earlier queue manager
Internally set properties are discarded. The MQRFH2 header is transferred unmodified. **PROPCTL** channel options are applied after internally set properties are discarded.

PUT Specifies whether messages can be put on the queue.

ENABLED
Messages can be added to the queue (by suitably authorized applications).

DISABLED
Messages cannot be added to the queue.

This parameter can also be changed using the MQSET API call.

QDEPTHHI(*integer***)**
The threshold against which the queue depth is compared to generate a Queue Depth High event.

This parameter is supported only on local and model queues.

This event indicates that an application put a message on a queue resulting in the number of messages on the queue becoming greater than or equal to the queue depth high threshold. See the **QDPHIEV** parameter.

The value is expressed as a percentage of the maximum queue depth (**MAXDEPTH** parameter), and must be in the range zero through 100 and no less than **QDEPTHLO**.

QDEPTHLO(*integer***)**
The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This parameter is supported only on local and model queues.

This event indicates that an application retrieved a message from a queue resulting in the number of messages on the queue becoming less than or equal to the queue depth low threshold. See the **QDPLOEV** parameter.

The value is expressed as a percentage of the maximum queue depth (**MAXDEPTH** parameter), and must be in the range zero through 100 and no greater than **QDEPTHHI**.

QDPHIEV
Controls whether Queue Depth High events are generated.

This parameter is supported only on local and model queues.

A Queue Depth High event indicates that an application put a message on a queue resulting in the number of messages on the queue becoming greater than or equal to the queue depth high threshold. See the **QDEPTHHI** parameter.

Note: The value of this parameter can change implicitly, and shared queues on z/OS affect the event. See the description of the Queue Depth High event in Queue Depth High.

ENABLED
Queue Depth High events are generated

DISABLED
Queue Depth High events are not generated

QDPLOEV
Controls whether Queue Depth Low events are generated.

This parameter is supported only on local and model queues.

A Queue Depth Low event indicates that an application retrieved a message from a queue resulting in the number of messages on the queue becoming less than or equal to the queue depth low threshold. See the **QDEPTHLO** parameter.

Note: The value of this parameter can change implicitly. For more information about this event, and the effect that shared queues on z/OS have on this event, see Queue Depth Low .

ENABLED
Queue Depth Low events are generated

DISABLED
Queue Depth Low events are not generated

QDPMAXEV
Controls whether Queue Full events are generated.

This parameter is supported only on local and model queues.

A Queue Full event indicates that a put to a queue was rejected because the queue is full. The queue depth reached its maximum value.

Note: The value of this parameter can change implicitly. For more information about this event, and the effect that shared queues on z/OS have on this event, see Queue Full.

ENABLED
Queue Full events are generated

DISABLED
Queue Full events are not generated

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object within the group.

Table 80. **QSGDISP** *parameters.*

Definitions of the **QSGDISP** parameters when defining a queue.

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the LIKE object. For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.

GROUP	The object definition resides in the shared repository but only if there is a shared queue manager environment. If the definition is successful, the following command is generated. The command is sent to all active queue managers to attempt to make or refresh local copies on page set zero:
	`DEFINE QUEUE(q-name)`
	`REPLACE QSGDISP(COPY)`
	
	The **DEFINE** command for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command. For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.
SHARED	This option applies only to local queues. The object is defined in the shared repository. Messages are stored in the coupling facility and are available to any queue manager in the queue-sharing group. You can specify SHARED only if: • CFSTRUCT is nonblank • INDXTYPE is not MSGTOKEN • The queue is not: – SYSTEM.CHANNEL.INITQ – SYSTEM.COMMAND.INPUT If the queue is clustered, a command is generated. The command is sent to all active queue managers in the queue-sharing group to notify them of this clustered, shared queue.

QSVCIEV

Controls whether Service Interval High or Service Interval OK events are generated.

This parameter is supported only on local and model queues and is ineffective if it is specified on a shared queue.

A Service Interval High event is generated when a check indicates that no messages were retrieved from the queue for at least the time indicated by the **QSVCINT** parameter.

A Service Interval OK event is generated when a check indicates that messages were retrieved from the queue within the time indicated by the **QSVCINT** parameter.

Note: The value of this parameter can change implicitly. For more information, see the description of the Service Interval High and Service Interval OK events in Queue Service Interval High and Queue Service Interval OK.

HIGH Service Interval High events are generated

OK Service Interval OK events are generated

NONE No service interval events are generated

QSVCINT(*integer***)**
The service interval used for comparison to generate Service Interval High and Service Interval OK events.

This parameter is supported only on local and model queues and is ineffective if it is specified on a shared queue.

See the **QSVCIEV** parameter.

The value is in units of milliseconds, and must be in the range zero through 999999999.

REPLACE & **NOREPLACE**
This option controls whether any existing definition, and on IBM WebSphere MQ for z/OS of the same disposition, is to be replaced with this one. Any object with a different disposition is not changed.

REPLACE
If the object does exist, the effect is like issuing the **ALTER** command without the **FORCE** parameter and with all the other parameters specified. In particular, note that any messages that are on the existing queue are retained.

There is a difference between the **ALTER** command without the **FORCE** parameter, and the **DEFINE** command with the **REPLACE** parameter. The difference is that **ALTER** does not change unspecified parameters, but **DEFINE** with **REPLACE** sets all the parameters. If you use **REPLACE**, unspecified parameters are taken either from the object named on the **LIKE** parameter, or from the default definition, and the parameters of the object being replaced, if one exists, are ignored.

The command fails if both of the following are true:

- The command sets parameters that would require the use of the **FORCE** parameter if you were using the ALTER command
- The object is open

The **ALTER** command with the **FORCE** parameter succeeds in this situation.

If **SCOPE(CELL)** is specified on UNIX and Linux systems, or Windows, and there is already a queue with the same name in the cell directory, the command fails, even if **REPLACE** is specified.

NOREPLACE
The definition must not replace any existing definition of the object.

RETINTVL(*integer***)**
The number of hours from when the queue was defined, after which the queue is no longer needed. The value must be in the range 0 - 999,999,999.

This parameter is supported only on local and model queues.

The CRDATE and CRTIME can be displayed using the **DISPLAY QUEUE** command.

This information is available for use by an operator or a housekeeping application to delete queues that are no longer required.

Note: The queue manager does not delete queues based on this value, nor does it prevent queues from being deleted if their retention interval is not expired. It is the responsibility of the user to take any required action.

RNAME(*string***)**
Name of remote queue. This parameter is the local name of the queue as defined on the queue manager specified by **RQMNAME**.

This parameter is supported only on remote queues.

- If this definition is used for a local definition of a remote queue, **RNAME** must not be blank when the open occurs.
- If this definition is used for a queue manager alias definition, **RNAME** must be blank when the open occurs.

  In a queue manager cluster, this definition applies only to the queue manager that made it. To advertise the alias to the whole cluster, add the **CLUSTER** attribute to the remote queue definition.
- If this definition is used for a reply-to queue alias, this name is the name of the queue that is to be the reply-to queue.

The name is not checked to ensure that it contains only those characters normally allowed for queue names; see Rules for naming IBM WebSphere MQ objects .

RQMNAME(*string***)**

The name of the remote queue manager on which the queue **RNAME** is defined.

This parameter is supported only on remote queues.

- If an application opens the local definition of a remote queue, **RQMNAME** must not be blank or the name of the local queue manager. When the open occurs, if **XMITQ** is blank there must be a local queue of this name, which is to be used as the transmission queue.
- If this definition is used for a queue manager alias, **RQMNAME** is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if **XMITQ** is blank, when the open occurs there must be a local queue of this name, which is to be used as the transmission queue.
- If **RQMNAME** is used for a reply-to queue alias, **RQMNAME** is the name of the queue manager that is to be the reply-to queue manager.

The name is not checked to ensure that it contains only those characters normally allowed for IBM WebSphere MQ object names; see Rules for naming IBM WebSphere MQ objects.

SCOPE

Specifies the scope of the queue definition.

This parameter is supported only on alias, local, and remote queues.

QMGR The queue definition has queue manager scope. This means that the definition of the queue does not extend beyond the queue manager that owns it. You can open a queue for output that is owned by another queue manager in either of two ways:

1. Specify the name of the owning queue manager.
2. Open a local definition of the queue on the other queue manager.

CELL The queue definition has cell scope. Cell scope means that the queue is known to all the queue managers in the cell. A queue with cell scope can be opened for output merely by specifying the name of the queue. The name of the queue manager that owns the queue need not be specified.

If there is already a queue with the same name in the cell directory, the command fails. The **REPLACE** option does not affect this situation.

This value is valid only if a name service supporting a cell directory is configured.

Restriction: The DCE name service is no longer supported.

This parameter is valid only on UNIX and Linux systems, and Windows.

SHARE and NOSHARE

Specifies whether multiple applications can get messages from this queue.

This parameter is supported only on local and model queues.

SHARE More than one application instance can get messages from the queue.

NOSHARE
Only a single application instance can get messages from the queue.

STATQ
Specifies whether statistics data collection is enabled:

QMGR Statistics data collection is based on the setting of the **STATQ** parameter of the queue manager.

ON If the value of the **STATQ** parameter of the queue manager is not NONE, statistics data collection for the queue is enabled.

OFF Statistics data collection for the queue is disabled.

If this parameter is used in an **ALTER** queue command, the change is effective only for connections to the queue manager made after the change to the parameter.

This parameter is valid only on IBM i, UNIX and Linux systems, and Windows.

STGCLASS(*string***)**
The name of the storage class.

This parameter is supported only on local and model queues.

This parameter is an installation-defined name.

This parameter is valid on z/OS only.

The first character of the name must be uppercase A through Z, and subsequent characters either uppercase A through Z or numeric 0 through 9.

Note: You can change this parameter only if the queue is empty and closed.

If you specify **QSGDISP**(SHARED) or **DEFTYPE**(SHAREDYN), this parameter is ignored.

TARGET(*string***)**
The name of the queue or topic object being aliased; See Rules for naming IBM WebSphere MQ objects . The object can be a queue or a topic as defined by **TARGTYPE**. The maximum length is 48 characters.

This parameter is supported only on alias queues.

This object needs to be defined only when an application process opens the alias queue.

This parameter is a synonym of the parameter **TARGQ**; **TARGQ** is retained for compatibility. If you specify **TARGET**, you cannot also specify **TARGQ**.

TARGTYPE(*string***)**
The type of object to which the alias resolves.

QUEUE The alias resolves to a queue.

TOPIC The alias resolves to a topic.

TRIGDATA(*string***)**
The data that is inserted in the trigger message. The maximum length of the string is 64 bytes.

This parameter is supported only on local and model queues.

For a transmission queue on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS, you can use this parameter to specify the name of the channel to be started.

This parameter can also be changed using the MQSET API call.

TRIGDPTH(*integer***)**
The number of messages that have to be on the queue before a trigger message is written, if **TRIGTYPE** is DEPTH. The value must be in the range 1 - 999,999,999.

This parameter is supported only on local and model queues.

This parameter can also be changed using the MQSET API call.

TRIGGER &**NOTRIGGER**
Specifies whether trigger messages are written to the initiation queue, named by the **INITQ** parameter, to trigger the application, named by the **PROCESS** parameter:

TRIGGER
Triggering is active, and trigger messages are written to the initiation queue.

NOTRIGGER
Triggering is not active, and trigger messages are not written to the initiation queue.

This parameter is supported only on local and model queues.

This parameter can also be changed using the MQSET API call.

TRIGMPRI(*integer***)**
The message priority number that triggers this queue. The value must be in the range zero through to the **MAXPRTY** queue manager parameter; see "DISPLAY QMGR" on page 672 for details.

This parameter can also be changed using the MQSET API call.

TRIGTYPE
Specifies whether and under what conditions a trigger message is written to the initiation queue. The initiation queue is (named by the **INITQ** parameter.

This parameter is supported only on local and model queues.

FIRST Whenever the first message of priority equal to or greater than the priority specified by the **TRIGMPRI** parameter of the queue arrives on the queue.

EVERY Every time a message arrives on the queue with priority equal to or greater than the priority specified by the **TRIGMPRI** parameter of the queue.

DEPTH When the number of messages with priority equal to or greater than the priority specified by **TRIGMPRI** is equal to the number indicated by the **TRIGDPTH** parameter.

NONE No trigger messages are written.

This parameter can also be changed using the MQSET API call.

USAGE
Queue usage.

This parameter is supported only on local and model queues.

NORMAL The queue is not a transmission queue.

XMITQ The queue is a transmission queue, which is used to hold messages that are destined for a remote queue manager. When an application puts a message to a remote queue, the message is stored on the appropriate transmission queue. It stays there, awaiting transmission to the remote queue manager.

If you specify this option, do not specify values for **CLUSTER** and **CLUSNL** and do not specify **INDXTYPE**(MSGTOKEN) or **INDXTYPE**(GROUPID).

XMITQ(*string***)**
The name of the transmission queue to be used for forwarding messages to the remote queue. **XMITQ** is used with either remote queue or queue manager alias definitions.

This parameter is supported only on remote queues.

If **XMITQ** is blank, a queue with the same name as **RQMNAME** is used as the transmission queue.

This parameter is ignored if the definition is being used as a queue manager alias and **RQMNAME** is the name of the local queue manager.

It is also ignored if the definition is used as a reply-to queue alias definition.

DEFINE QALIAS:

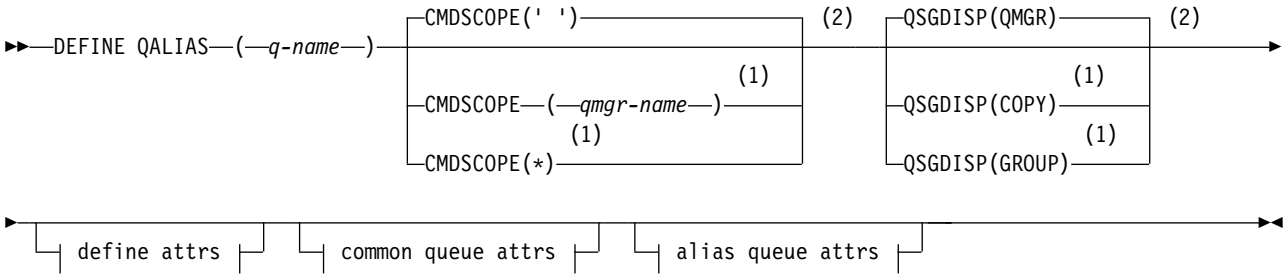
Use DEFINE QALIAS to define a new alias queue, and set its parameters.

Note: An alias queue provides a level of indirection to another queue or a topic object. If the alias refers to a queue, it must be another local or remote queue, defined at this queue manager, or a clustered alias queue defined on another queue manager. It cannot be another alias queue on this queue manager. If the alias refers to a topic, it must be a topic object defined at this queue manager.

- Syntax diagram
- "Usage notes for DEFINE queues" on page 511
- "Parameter descriptions for DEFINE QUEUE and ALTER QUEUE" on page 512

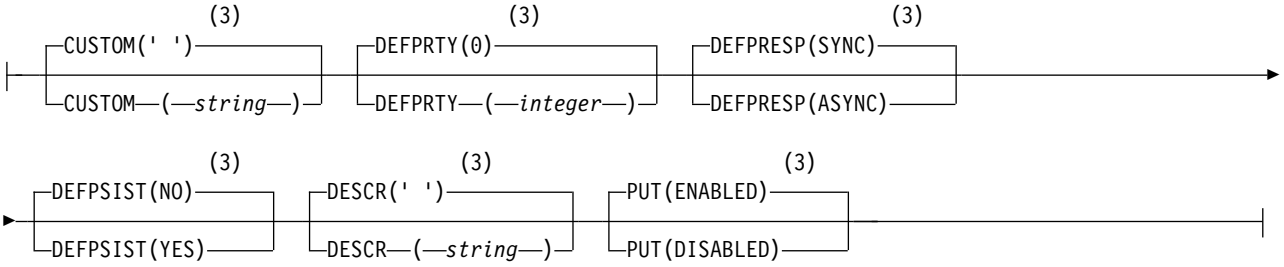
Synonym: DEF QA

DEFINE QALIAS

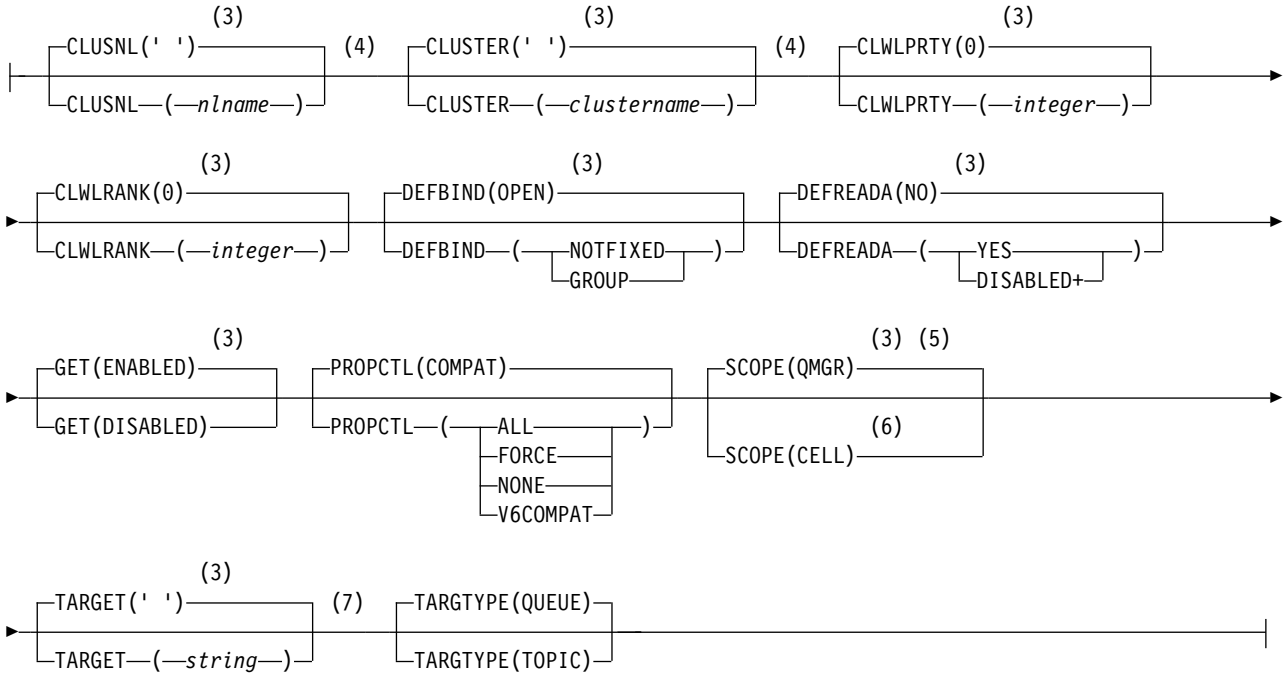


Define attrs:

Common queue attrs:



Alias queue attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 On z/OS, the QALIAS name can only be the same as the TARGQ name if the target queue is a cluster queue.

3 This is the default supplied with IBM WebSphere MQ, but your installation might have changed it.

4 Valid only on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS.

5 Valid only on IBM i, UNIX and Linux systems, and Windows.

6 Valid only on UNIX and Linux systems, and Windows.

7 The TARGQ parameter is available for compatibility with previous releases. It is a synonym of TARGET; you cannot specify both parameters.

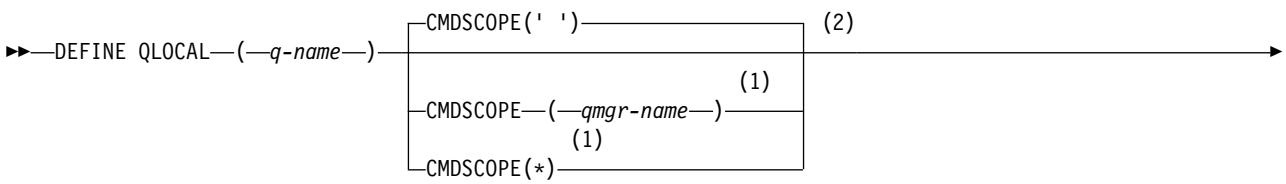
DEFINE QLOCAL:

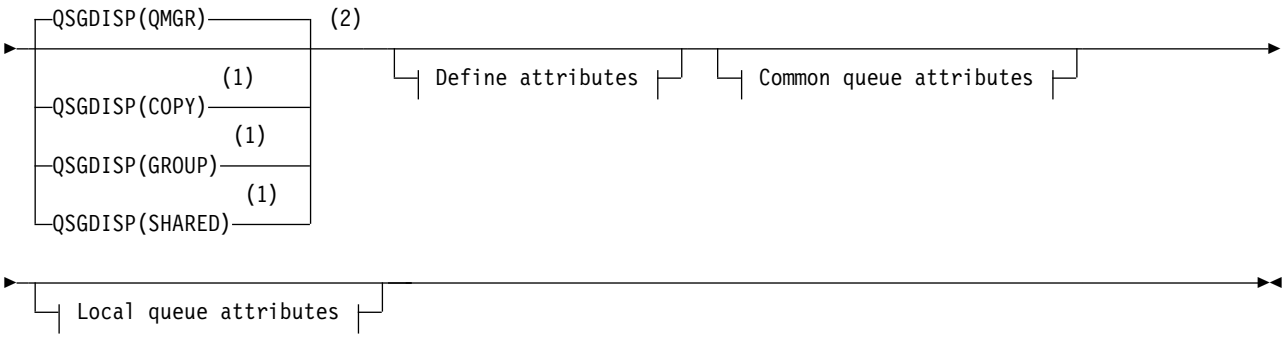
Use **DEFINE QLOCAL** to define a new local queue, and set its parameters.

- Syntax diagram
- "Usage notes for DEFINE queues" on page 511
- "Parameter descriptions for DEFINE QUEUE and ALTER QUEUE" on page 512

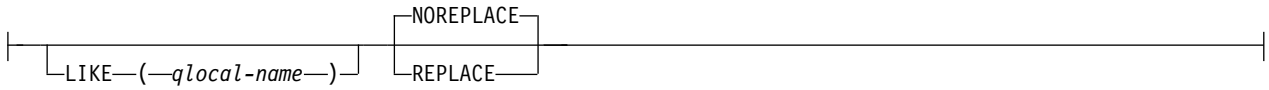
Synonym: **DEF QL**

DEFINE QLOCAL

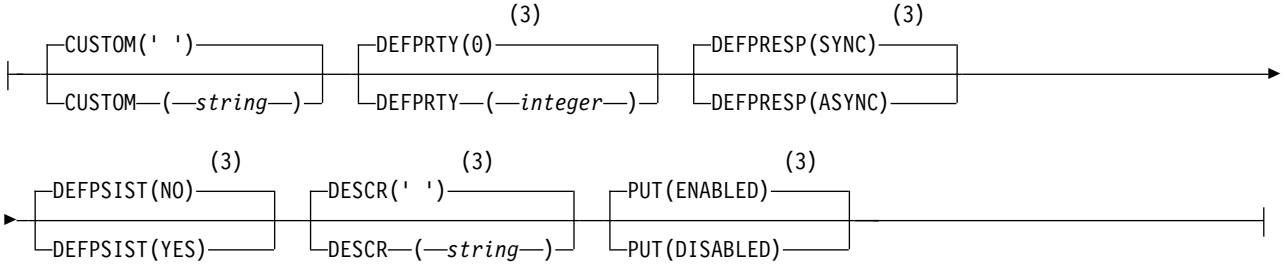




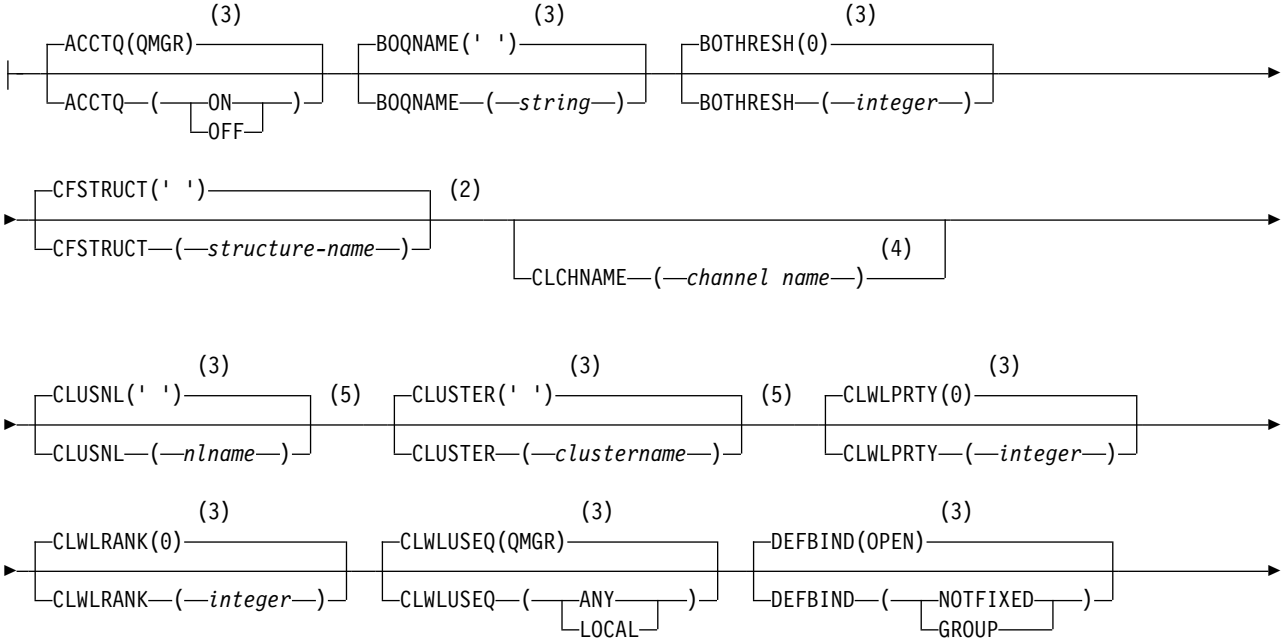
Define attributes:

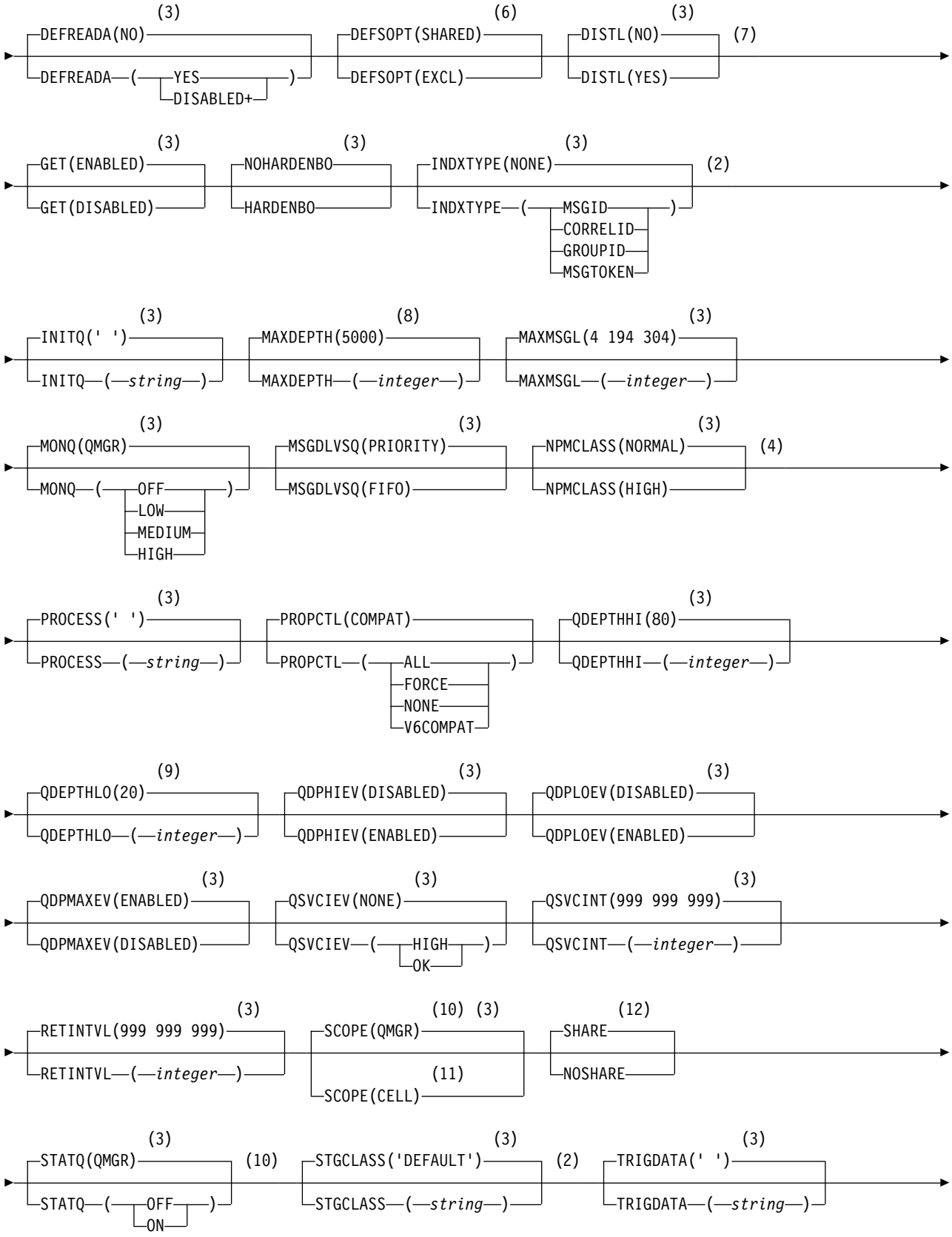


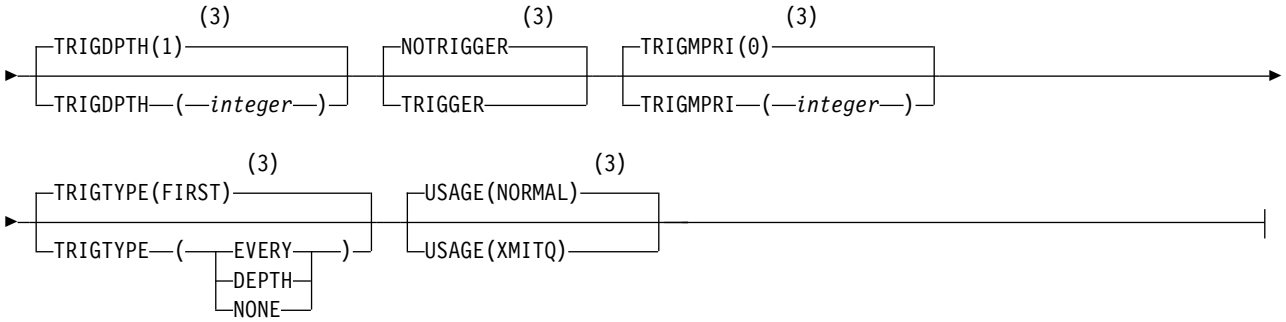
Common queue attributes:



Local queue attributes:




(3)
DEFREADA(NO)
DEFREADA(YES)
DISABLED+
(6)
DEFSOPT(SHARED)
DEFSOPT(EXCL)
(3)
DISTL(NO)
DISTL(YES)
(7)
(3)
GET(ENABLED)
GET(DISABLED)
(3)
NOHARDENBO
HARDENBO
(3)
INDXTYPE(NONE)
INDXTYPE(MSGID
CORRELID
GROUPID
MSGTOKEN)
(2)
(3)
INITQ(' ')
INITQ(string)
(8)
MAXDEPTH(5000)
MAXDEPTH(integer)
(3)
MAXMSGL(4 194 304)
MAXMSGL(integer)
(3)
MONQ(QMGR)
MONQ(OFF
LOW
MEDIUM
HIGH)
(3)
MSGDLVSQ(PRIORITY)
MSGDLVSQ(FIFO)
(3)
NPMCLASS(NORMAL)
NPMCLASS(HIGH)
(4)
(3)
PROCESS(' ')
PROCESS(string)
PROPCTL(COMPAT)
PROPCTL(ALL
FORCE
NONE
V6COMPAT)
(3)
QDEPTHHI(80)
QDEPTHHI(integer)
(9)
QDEPTHLO(20)
QDEPTHLO(integer)
(3)
QDPHIEV(DISABLED)
QDPHIEV(ENABLED)
(3)
QDPLOEV(DISABLED)
QDPLOEV(ENABLED)
(3)
QDPMAXEV(ENABLED)
QDPMAXEV(DISABLED)
(3)
QSVCIEV(NONE)
QSVCIEV(HIGH
OK)
(3)
QSVCINT(999 999 999)
QSVCINT(integer)
(3)
RETINTVL(999 999 999)
RETINTVL(integer)
(10) (3)
SCOPE(QMGR)
(11)
SCOPE(CELL)
(12)
SHARE
NOSHARE
(3)
STATQ(QMGR)
STATQ(OFF
ON)
(10)
(3)
STGCLASS('DEFAULT')
STGCLASS(string)
(2)
(3)
TRIGDATA(' ')
TRIGDATA(string)



Notes:

1 Valid only on z/OS and when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 This is the default supplied with IBM WebSphere MQ, but your installation might have changed it.

4 Not valid on z/OS.

5 Valid on UNIX, Linux, IBM i, Windows, and z/OS systems.

6 This is the default supplied with IBM WebSphere MQ (except on z/OS, where it is EXCL), but your installation might have changed it.

7 Valid on IBM i, UNIX, Linux, and Windows systems.

8 This is the default supplied with IBM WebSphere MQ (except on z/OS, where it is 999 999 999), but your installation might have changed it.

9 This is the default supplied with IBM WebSphere MQ (except on z/OS where it is 40), but your installation might have changed it.

10 Valid on IBM i, UNIX, Linux, and Windows systems.

11 Valid only on UNIX, Linux, and Windows systems.

12 This is the default supplied with IBM WebSphere MQ (except on z/OS, where it is NOSHARE), but your installation might have changed it.

DEFINE QMODEL:

Use **DEFINE QMODEL** to define a new model queue, and set its parameters.

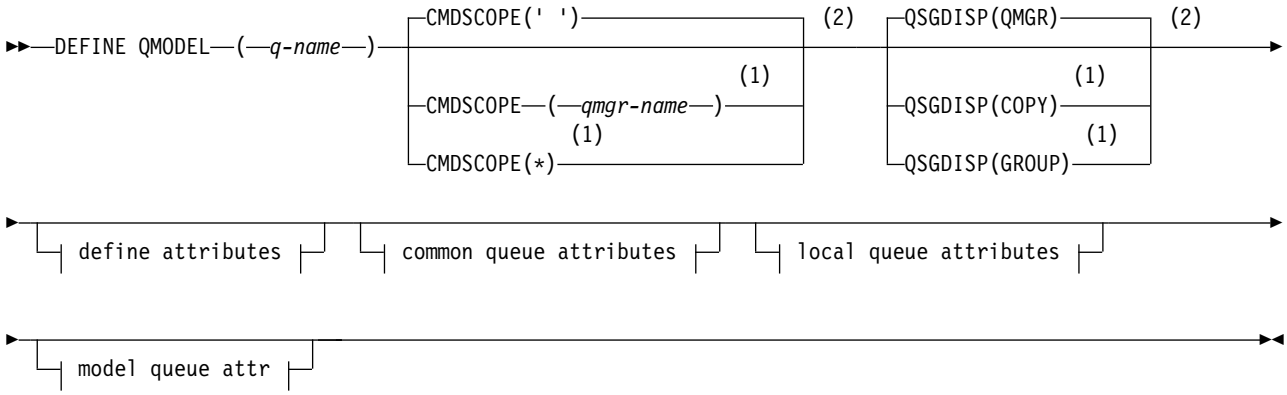
A model queue is not a real queue, but a collection of attributes that you can use when creating dynamic queues with the MQOPEN API call.

When it has been defined, a model queue (like any other queue) has a complete set of applicable attributes, even if some of these are defaults.

- Syntax diagram
- "Usage notes for DEFINE queues" on page 511
- "Parameter descriptions for DEFINE QUEUE and ALTER QUEUE" on page 512

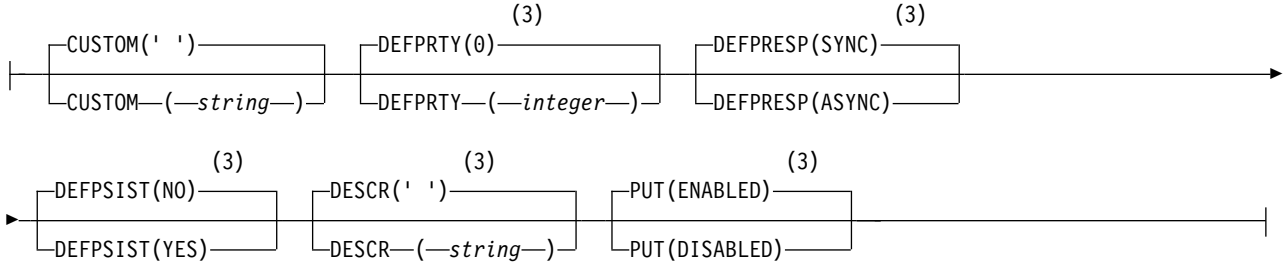
Synonym: **DEF QM**

DEFINE QMODEL

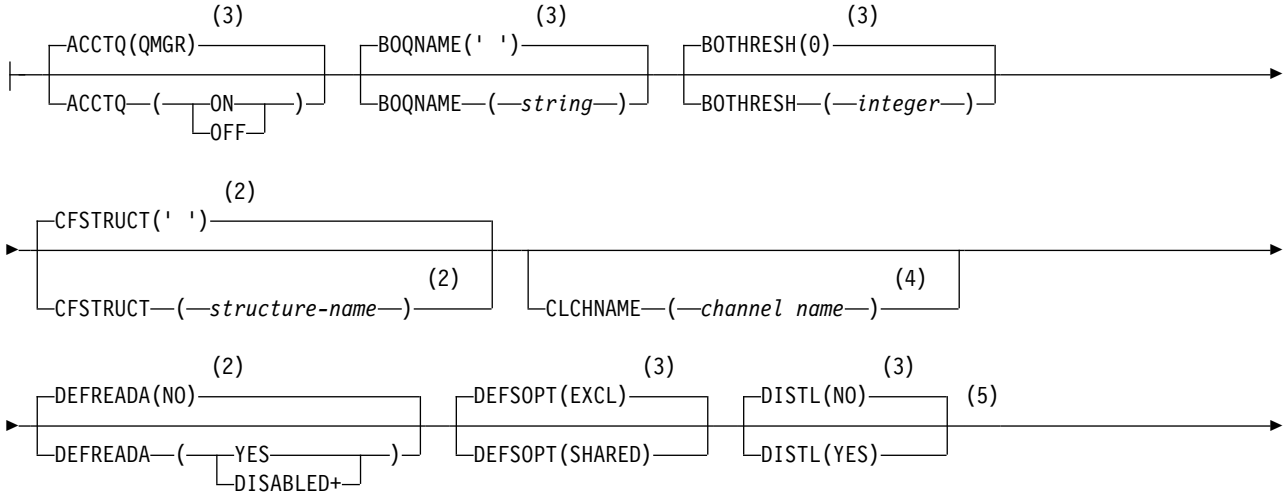


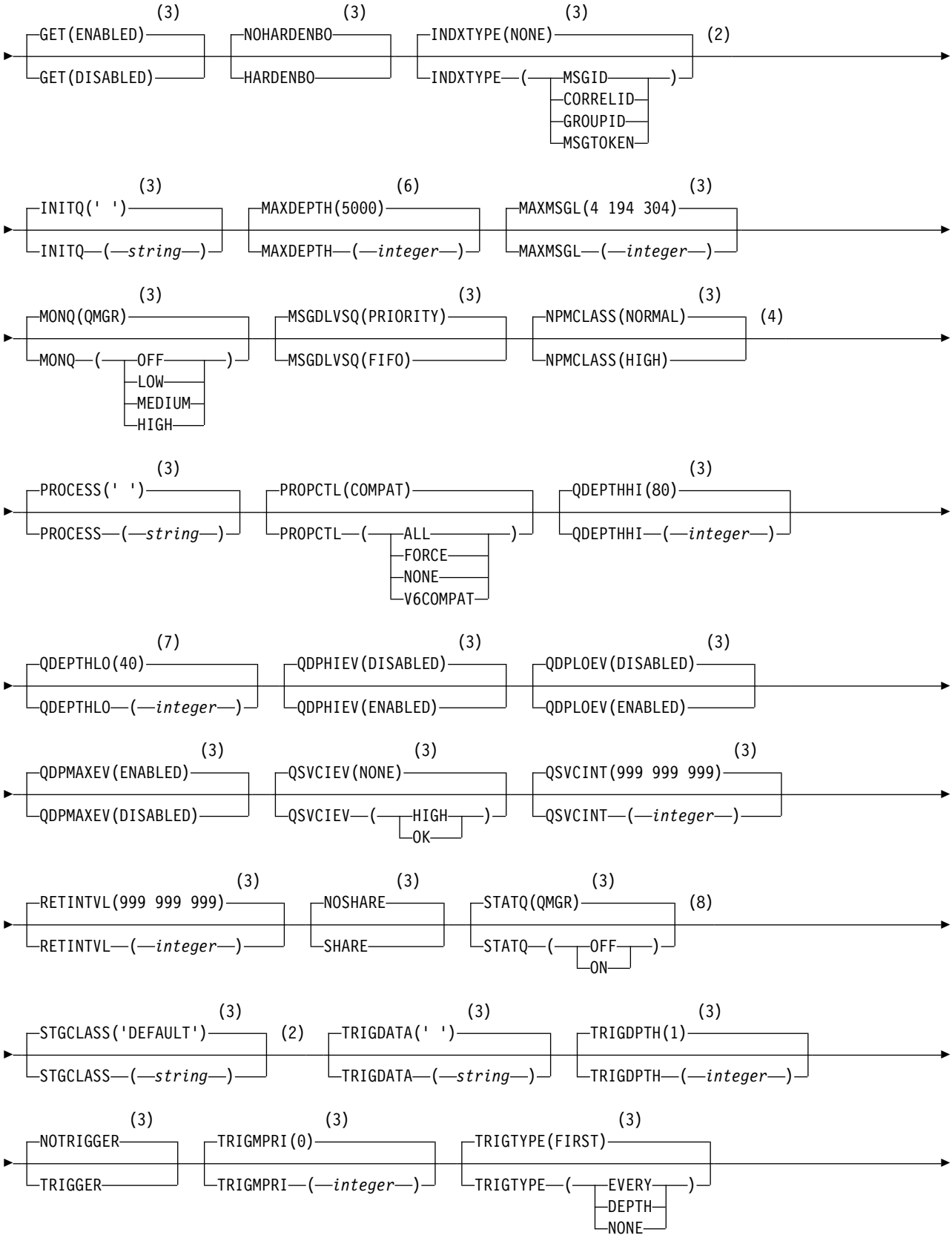
Define attributes:

Common queue attributes:



Local queue attributes:




(3)
GET(ENABLED)
GET(DISABLED)
(3)
NOHARDENBO
HARDENBO
(3)
INDXTYPE(NONE)
INDXTYPE—(—MSGID—CORRELID—GROUPID—MSGTOKEN—)
(2)
(3)
INITQ(' ')
INITQ—(—string—)
(6)
MAXDEPTH(5000)
MAXDEPTH—(—integer—)
(3)
MAXMSGL(4 194 304)
MAXMSGL—(—integer—)
(3)
MONQ(QMGR)
MONQ—(—OFF—LOW—MEDIUM—HIGH—)
(3)
MSGDLVSQ(PRIORITY)
MSGDLVSQ(FIFO)
(3)
NPMCLASS(NORMAL)
NPMCLASS(HIGH)
(4)
(3)
PROCESS(' ')
PROCESS—(—string—)
PROPCTL(COMPAT)
PROPCTL—(—ALL—FORCE—NONE—V6COMPAT—)
(3)
QDEPTHHI(80)
QDEPTHHI—(—integer—)
(7)
QDEPTHLO(40)
QDEPTHLO—(—integer—)
(3)
QDPHIEV(DISABLED)
QDPHIEV(ENABLED)
(3)
QDPLOEV(DISABLED)
QDPLOEV(ENABLED)
(3)
QDPMAXEV(ENABLED)
QDPMAXEV(DISABLED)
(3)
QSVCIEV(NONE)
QSVCIEV—(—HIGH—OK—)
(3)
QSVCINT(999 999 999)
QSVCINT—(—integer—)
(3)
RETINTVL(999 999 999)
RETINTVL—(—integer—)
(3)
NOSHARE
SHARE
(3)
STATQ(QMGR)
STATQ—(—OFF—ON—)
(8)
(3)
STGCLASS('DEFAULT')
STGCLASS—(—string—)
(2)
(3)
TRIGDATA(' ')
TRIGDATA—(—string—)
(3)
TRIGDPTH(1)
TRIGDPTH—(—integer—)
(3)
NOTRIGGER
TRIGGER
(3)
TRIGMPRI(0)
TRIGMPRI—(—integer—)
(3)
TRIGTYPE(FIRST)
TRIGTYPE—(—EVERY—DEPTH—NONE—)

Model queue attr:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Used only on z/OS.

3 This is the default supplied with WebSphere MQ, but your installation might have changed it.

4 Not valid on z/OS.

5 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

6 This is the default supplied with WebSphere MQ (except on z/OS, where it is 999 999 999), but your installation might have changed it.

7 This is the default supplied with WebSphere MQ (except on platforms other than z/OS where it is 20), but your installation might have changed it.

8 Valid only on IBM i, UNIX systems, and Windows.

DEFINE QREMOTE:

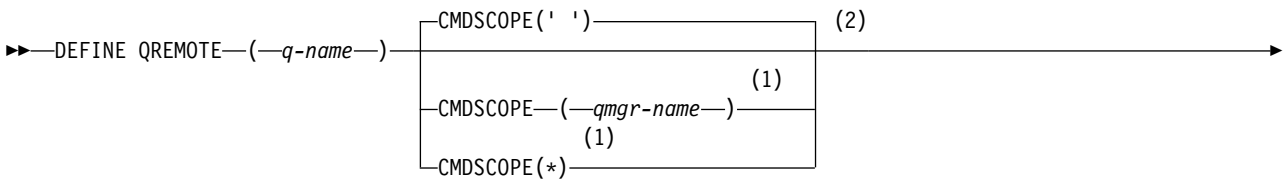
Use DEFINE QREMOTE to define a new local definition of a remote queue, a queue manager alias, or a reply-to queue alias, and to set its parameters.

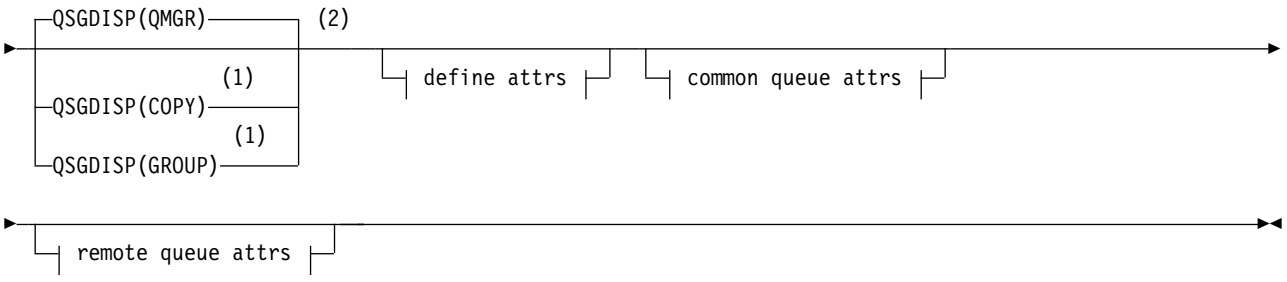
A remote queue is one that is owned by another queue manager that application processes connected to this queue manager need to access.

- Syntax diagram
- "Usage notes for DEFINE queues" on page 511
- "Parameter descriptions for DEFINE QUEUE and ALTER QUEUE" on page 512

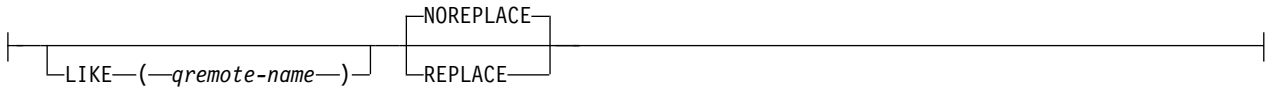
Synonym: DEF QR

DEFINE QREMOTE

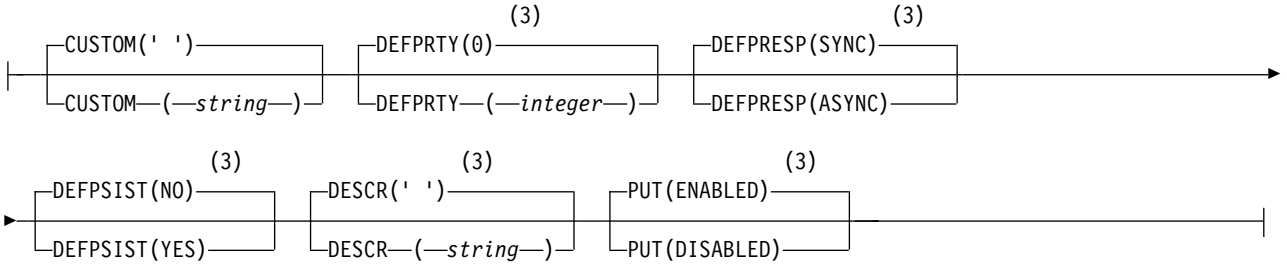




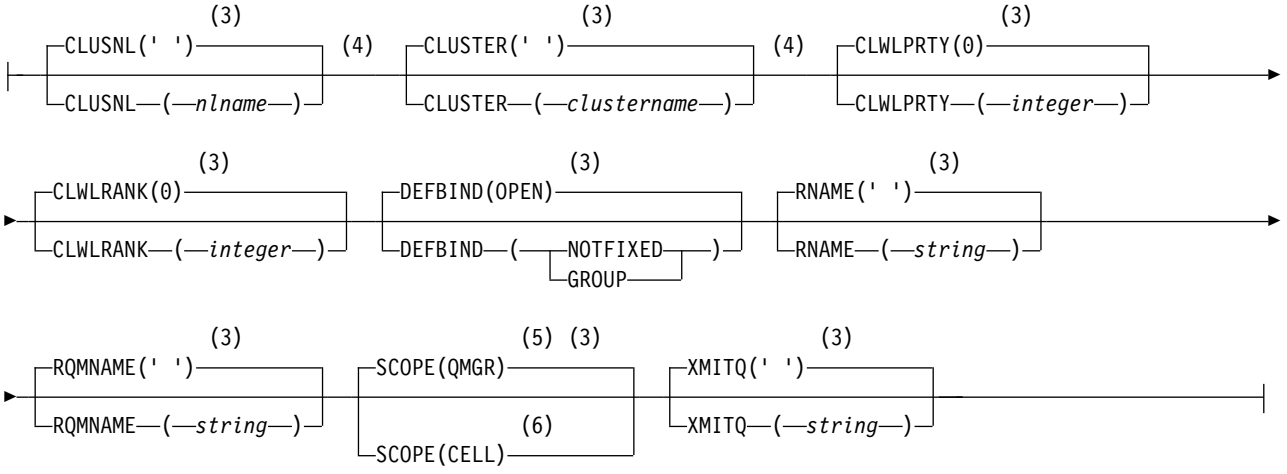
Define attrs:



Common queue attrs:



Remote queue attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 This is the default supplied with IBM WebSphere MQ, but your installation might have changed it.

4 Valid only on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS.

5 Valid only on IBM i, UNIX and Linux systems, and Windows.

6 Valid only on UNIX and Linux systems, and Windows.

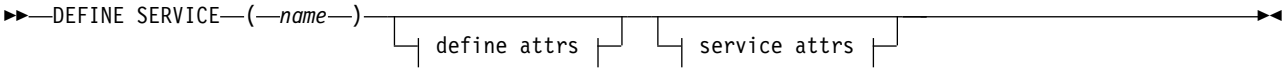
DEFINE SERVICE:

Use the MQSC command DEFINE SERVICE to define a new WebSphere MQ service definition, and set its parameters.

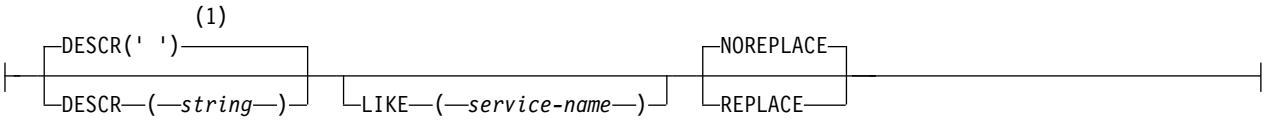
UNIX and Linux	Windows
✓	✓

Synonym:

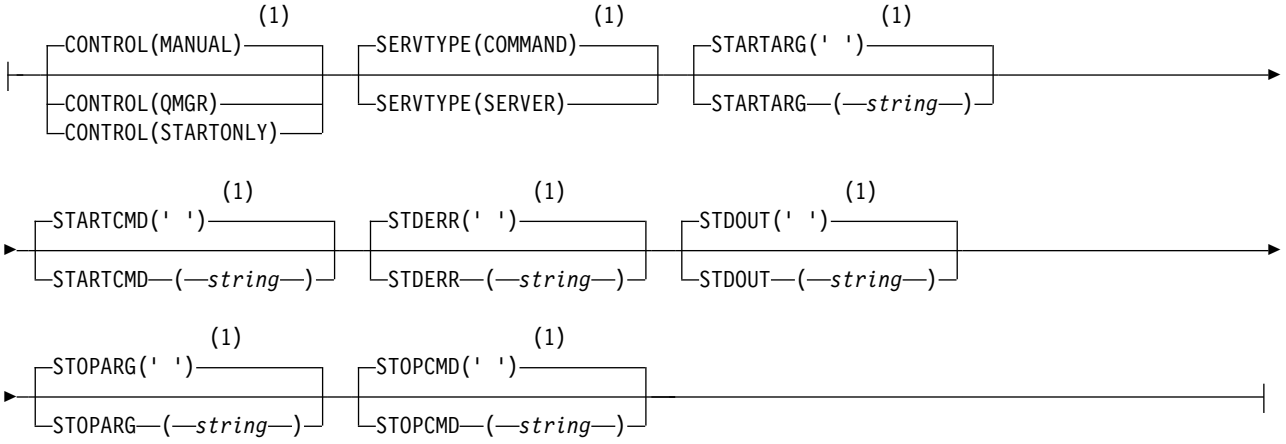
DEFINE SERVICE



Define attrs:



Service attrs:



Notes:

1 This is the default supplied with WebSphere MQ, but your installation might have changed it.

Usage notes

A service is used to define the user programs that are to be started and stopped when the queue manager is started and stopped. You can also start and stop these programs by issuing the START SERVICE and STOP SERVICE commands.

Attention: This command allows a user to run an arbitrary command with mqm authority. If granted rights to use this command, a malicious or careless user could define a service which damages your systems or data, for example, by deleting essential files.

For more information about services, see Services.

Parameter descriptions for DEFINE SERVICE

The parameter descriptions apply to the ALTER SERVICE and DEFINE SERVICE commands, with the following exceptions:

- The **LIKE** parameter applies only to the DEFINE SERVICE command.
- The **NOREPLACE** and **REPLACE** parameter applies only to the DEFINE SERVICE command.

(service-name)
Name of the WebSphere MQ service definition (see Rules for naming IBM WebSphere MQ objects).

The name must not be the same as any other service definition currently defined on this queue manager (unless REPLACE is specified).

CONTROL(*string***)**
Specifies how the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the START SERVICE and STOP SERVICE commands.

QMGR
The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the service when an operator issues the DISPLAY SERVICE command (see "DISPLAY SERVICE" on page 717).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

LIKE(*service-name***)**
The name of a service the parameters of which are used to model this definition.

This parameter applies only to the DEFINE SERVICE command.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from the default definition for services on this queue manager. Not completing this parameter is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.SERVICE)
```

A default service is provided but it can be altered by the installation of the default values required. See Rules for naming IBM WebSphere MQ objects .

REPLACE and NOREPLACE
Whether the existing definition is to be replaced with this one.

This parameter applies only to the DEFINE SERVICE command.

REPLACE
The definition must replace any existing definition of the same name. If a definition does not exist, one is created.

NOREPLACE
The definition should not replace any existing definition of the same name.

SERVTYPE
Specifies the mode in which the service is to run:

COMMAND
A command service object. Multiple instances of a command service object can be executed concurrently. You cannot monitor the status of command service objects.

SERVER
A server service object. Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the DISPLAY SVSTATUS command.

STARTARG(*string***)**
Specifies the arguments to be passed to the user program at queue manager startup.

STARTCMD(*string***)**
Specifies the name of the program which is to run. You must specify a fully qualified path name to the executable program.

STDERR(*string***)**
Specifies the path to a file to which the standard error (stderr) of the service program is redirected. If the file does not exist when the service program is started, the file is created. If this value is blank then any data written to stderr by the service program is discarded.

STDOUT(*string***)**
Specifies the path to a file to which the standard output (stdout) of the service program is redirected. If the file does not exist when the service program is started, the file is created. If this value is blank then any data written to stdout by the service program is discarded.

STOPARG(*string***)**
Specifies the arguments to be passed to the stop program when instructed to stop the service.

STOPCMD(*string***)**
Specifies the name of the executable program to run when the service is requested to stop. You must specify a fully qualified path name to the executable program.

Replaceable inserts can be used for any of the STARTCMD, STARTARG, STOPCMD, STOPARG, STDOUT or STDERR strings, for more information, see Replaceable inserts on service definitions.

Related information:
Working with services

DEFINE SUB:

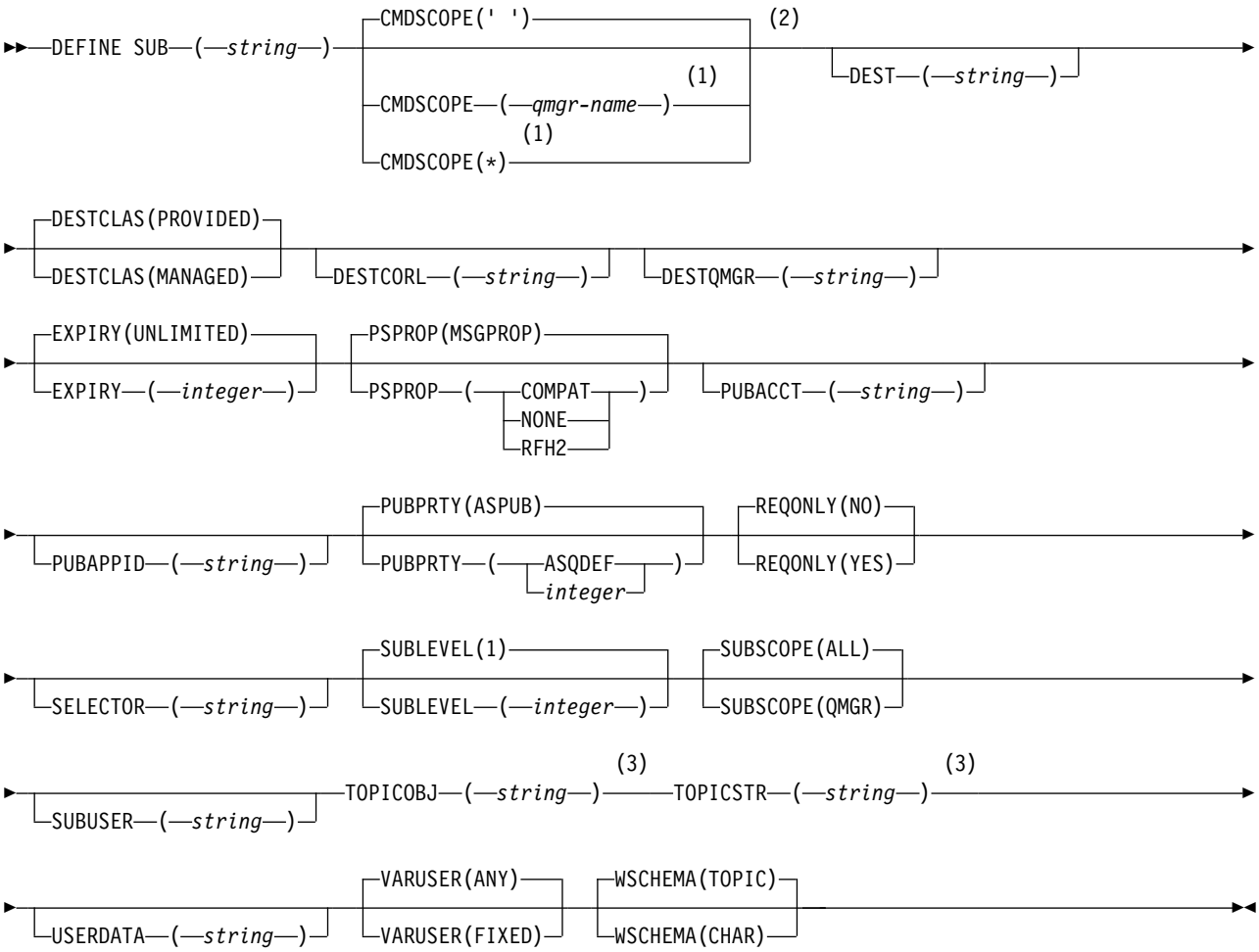
Use DEFINE SUB to allow an existing application to participate in a publish/subscribe application by allowing the administrative creation of a durable subscription.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes for DEFINE SUB" on page 548
- "Parameter descriptions for DEFINE SUB" on page 548

Synonym: DEF SUB

DEFINE SUB



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 At least one of **TOPICSTR** and **TOPICOBJ** must be present on **DEFINE**.

Usage notes for DEFINE SUB

1. You must provide the following information when you define a subscription:
   - The SUBNAME
   - A destination for messages
   - The topic to which the subscription applies
2. You can provide the topic name in the following ways:

   TOPICSTR
   The topic is fully specified as the TOPICSTR attribute.

   TOPICOBJ
   The topic is obtained from the TOPICSTR attribute of the named topic object. The named topic object is retained as the TOPICOBJ attribute of the new subscription. This method is provided to help you enter long topic strings through an object definition.

   TOPICSTR and TOPICOBJ
   The topic is obtained by the concatenation of the TOPICSTR attribute of the named topic object and the value of TOPICSTR (see the MQSUB API specification for concatenation rules). The named topic object is retained as the TOPICOBJ attribute of the new subscription.
3. If you specify TOPICOBJ, the parameter must name a WebSphere MQ topic object. The existence of the named topic object is checked at the time the command processes.
4. You can explicitly specify the destination for messages through the use of the DEST and DESTQMGR keywords.

   You must provide the DEST keyword for the default option of DESTCLAS(PROVIDED); if you specify DESTCLAS(MANAGED), a managed destination is created on the local queue manager, so you cannot specify either the DEST or DESTQMGR attribute.
5. On z/OS only, at the time the DEF SUB command processes, no check is performed that the named DEST or DESTQMGR exists.

   These names are used at publishing time as the *ObjectName* and *ObjectQMgrName* for an MQOPEN call. These names are resolved according to the WebSphere MQ name resolution rules.
6. When a subscription is defined administratively using MQSC or PCF commands, the selector is not validated for invalid syntax. The DEFINE SUB command has no equivalent to the MQRC_SELECTION_NOT_AVAILABLE reason code that can be returned by the MQSUB API call.
7. TOPICOBJ, TOPICSTR, WSCHEMA, SELECTOR, SUBSCOPE, and DESTCLAS cannot be changed with DEFINE REPLACE.
8. When a publication has been retained, it is no longer available to subscribers at higher levels because it is republished at PubLevel 1.

Parameter descriptions for DEFINE SUB

(string)
A mandatory parameter. Specifies the unique name for this subscription, see **SUBNAME** property.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is processed when the queue manager is a member of a queue-sharing group.

'' The command is processed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is processed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of setting this value is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

DEST(*string***)**
The destination for messages published to this subscription; this parameter is the name of a queue.

DESTCLAS
System managed destination.

PROVIDED
The destination is a queue.

MANAGED
The destination is managed.

DESTCORL(*string***)**
The *CorrelId* used for messages published to this subscription.

DESTQMGR(*string***)**
The destination queue manager for messages published to this subscription. You must define the channels to the remote queue manager, for example, the XMITQ, and a sender channel. If you do not, messages do not arrive at the destination.

EXPIRY
The time to expiry of the subscription object from the creation date and time.

(*integer***)**
The time to expiry, in tenths of a second, from the creation date and time.

UNLIMITED
There is no expiry time. This is the default option supplied with the product.

LIKE(*subscription-name***)**

The name of a subscription, the parameters of which are used as a model for this definition.

This parameter applies only to the DEFINE SUB command.

If this field is not supplied, and you do not complete the parameter fields related to the command, the values are taken from the default definition for subscriptions on this queue manager. Not completing this parameter is equivalent to specifying:

```
LIKE (SYSTEM.DEFAULT.SUB)
```

PSPROP
The manner in which publish subscribe related message properties are added to messages sent to this subscription.

NONE
Do not add publish subscribe properties to the message.

COMPAT
Publish subscribe properties are added within an MQRFH version 1 header unless the message was published in PCF format.

MSGPROP
Publish subscribe properties are added as message properties.

RFH2 Publish subscribe properties are added within an MQRFH version 2 header.

PUBACCT(*string***)**
Accounting token passed by the subscriber, for propagation into messages published to this subscription in the *AccountingToken* field of the MQMD.

PUBAPPID(*string***)**
Identity data passed by the subscriber, for propagation into messages published to this subscription in the *ApplIdentityData* field of the MQMD.

PUBPRTY
The priority of the message sent to this subscription.

ASPUB
Priority of the message sent to this subscription is taken from the priority supplied in the published message.

ASQDEF
Priority of the message sent to this subscription is taken from the default priority of the queue defined as a destination.

(integer)
An integer providing an explicit priority for messages published to this subscription.

REPLACE and NOREPLACE
This parameter controls whether any existing definition is to be replaced with this one.

REPLACE
The definition replaces any existing definition of the same name. If a definition does not exist, one is created.

You cannot change TOPICOBJ, TOPICSTR, WSCHEMA, SELECTOR, SUBSCOPE, or DESTCLAS with DEFINE REPLACE.

NOREPLACE
The definition does not replace any existing definition of the same name.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.

NO All publications on the topic are delivered to this subscription.

YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

This parameter is equivalent to the subscribe option MQSO_PUBLICATIONS_ON_REQUEST.

SELECTOR(*string***)**
A selector that is applied to messages published to the topic.

SUBLEVEL(*integer***)**
The level within the subscription hierarchy at which this subscription is made. The range is zero through 9.

SUBSCOPE
Determines whether this subscription is forwarded to other queue managers, so that the subscriber receives messages published at those other queue managers.

ALL The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy.

QMGR
The subscription forwards messages published on the topic only within this queue manager.

Note: Individual subscribers can only *restrict* **SUBSCOPE**. If the parameter is set to ALL at topic level, then an individual subscriber can restrict it to QMGR for this subscription. However, if the parameter is set to QMGR at topic level, then setting an individual subscriber to ALL has no effect.

SUBNAME
The application's unique subscription name that is associated with the handle. This parameter is relevant only for handles of subscriptions to topics. It is not returned for other handles. Not all subscriptions will have a subscription name.

SUBUSER(*string***)**
Specifies the user ID that is used for security checks that are performed to ensure that publications can be put to the destination queue associated with the subscription. This ID is either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription. The length of this parameter must not exceed 12 characters.

TOPICOBJ(*string***)**
The name of a topic object used by this subscription.

TOPICSTR(*string***)**
Specifies a fully qualified topic name, or a topic set using wildcard characters for the subscription.

USERDATA(*string***)**
Specifies the user data associated with the subscription. The string is a variable length value that can be retrieved by the application on an MQSUB API call and passed in a message sent to this subscription as a message property.

V7.5.0.8 From Version 7.5.0, Fix Pack 8, an IBM WebSphere MQ classes for JMS application can retrieve the subscription user data from the message by using the constant JMS_IBM_SUBSCRIPTION_USER_DATA in the JmsConstants interface with the method javax.jms.Message.getStringProperty(java.lang.String). For more information, see Retrieval of user subscription data.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.

ANY Any user can connect to and takeover ownership of the subscription.

FIXED
Takeover by another **USERID** is not permitted.

WSCHEMA
The schema to be used when interpreting any wildcard characters in the topic string.

CHAR
Wildcard characters represent portions of strings.

TOPIC
Wildcard characters represent portions of the topic hierarchy.

DEFINE TOPIC:

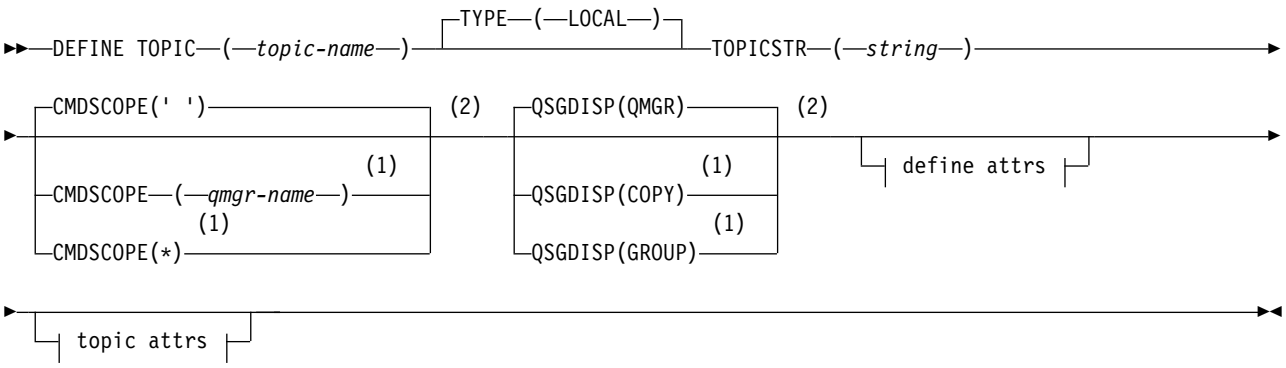
Use DEFINE TOPIC to define a new WebSphere MQ administrative topic in a topic tree, and set its parameters.

UNIX and Linux	Windows
✓	✓

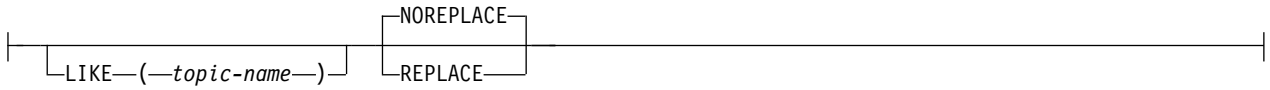
- Syntax diagram
- "Usage notes for DEFINE TOPIC" on page 553
- "Parameter descriptions for DEFINE TOPIC" on page 554

Synonym: DEF TOPIC

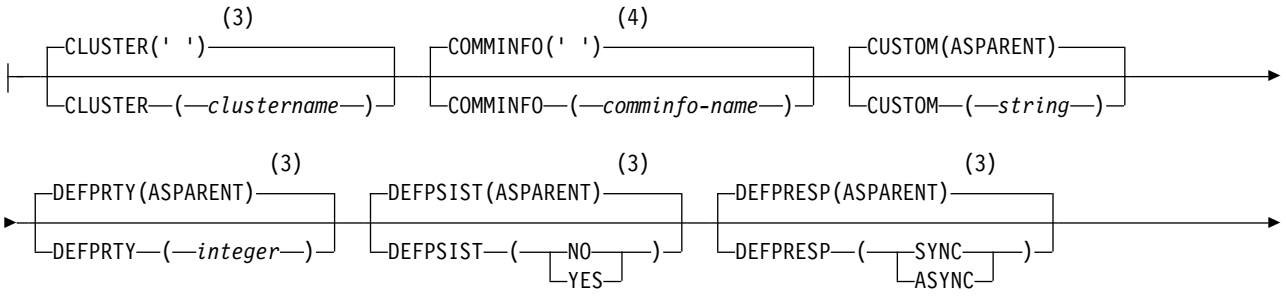
DEFINE TOPIC

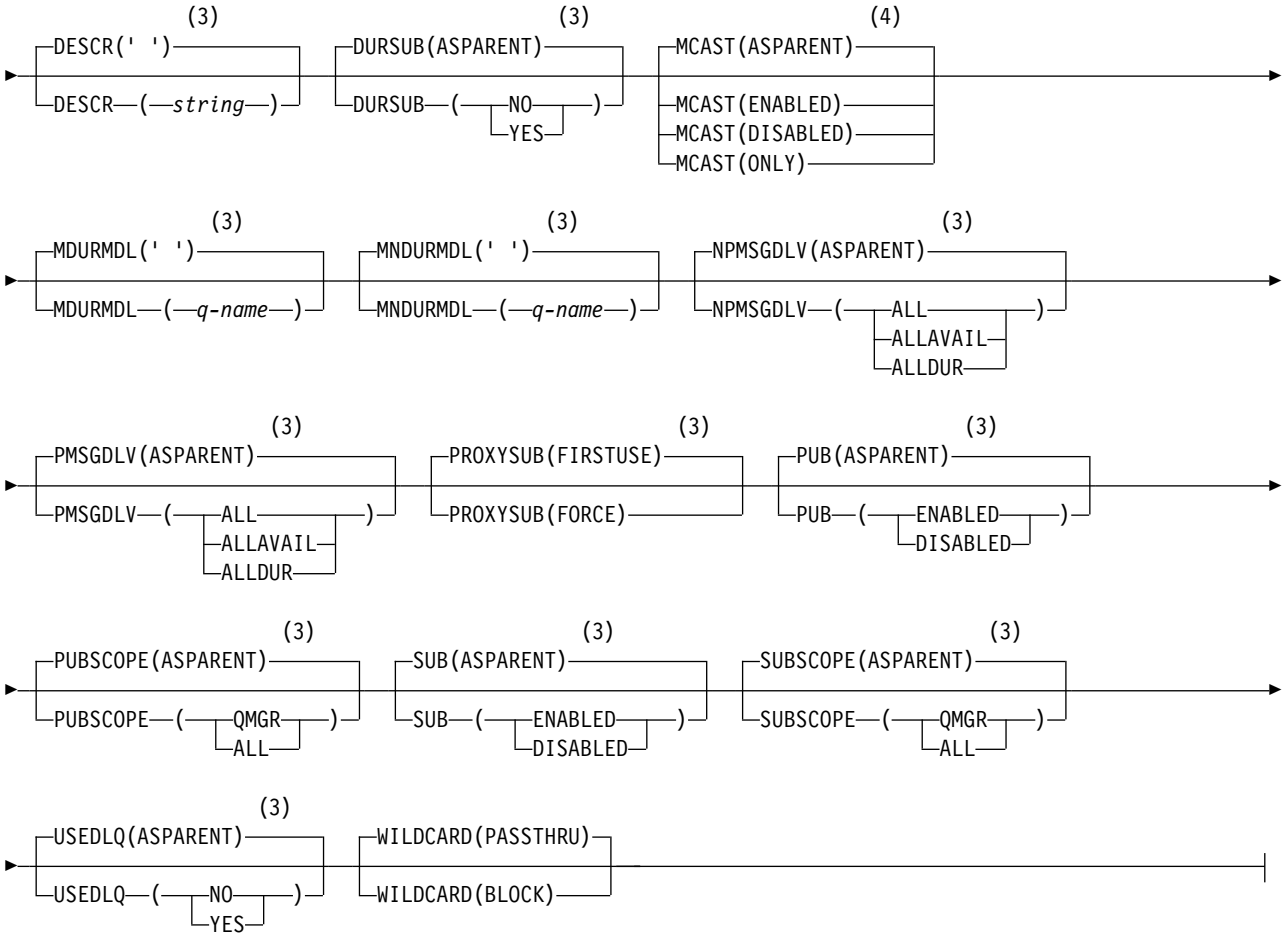


Define attrs:



Topic attrs:





Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 This is the default supplied with WebSphere MQ, but your installation might have changed it.

4 Not valid on z/OS.

Usage notes for DEFINE TOPIC

- When an attribute has the value ASPARENT, the value is taken from the setting of the first parent administrative node that is found in the topic tree. Administered nodes are based on either locally defined topic objects or remotely defined cluster topics when participating in a publish/subscribe cluster. If the first parent topic object also has the value ASPARENT, the next object is looked for. If every object that is found, when looking up the tree, uses ASPARENT, the values are taken from the SYSTEM.BASE.TOPIC, if it exists. If SYSTEM.BASE.TOPIC does not exist, the values are the same as the values supplied with WebSphere MQ in the definition of the SYSTEM.BASE.TOPIC.
- The ASPARENT attribute is applied at each queue manager in the cluster collective by inspecting the set of local definitions and cluster definitions that is visible in the queue manager at the time.
- When a publication is sent to multiple subscribers, the attributes used from the topic object are used consistently for all subscribers that receive the publication. For example, inhibiting publication on a topic is applied for the next application MQPUT to the topic. A publication that is in progress to multiple subscribers completes to all subscribers. This publication does not take note of a change that has happened, part of the way through, to any attribute on the topic.

Parameter descriptions for DEFINE TOPIC

(topic-name)
Name of the WebSphere MQ topic definition (see Rules for naming IBM WebSphere MQ objects). The maximum length is 48 characters.

The name must not be the same as any other topic definition currently defined on this queue manager (unless REPLACE is specified).

CLUSTER
The name of the cluster to which this topic belongs. Setting this parameter to a cluster that this queue manager is a member of makes all queue managers in the cluster aware of this topic. Any publication to this topic or a topic string below it put to any queue manager in the cluster is propagated to subscriptions on any other queue manager in the cluster. For more details, see Distributed publish/subscribe.

'' If no topic object above this topic in the topic tree has set this parameter to a cluster name, then this topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers. If a topic node higher in the topic tree has a cluster name set, publications and subscriptions to this topic are also propagated throughout the cluster.

string The topic belongs to this cluster. It is not recommended that this is set to a different cluster from a topic object above this topic object in the topic tree. Other queue managers in the cluster will honour this object's definition unless a local definition of the same name exists on those queue managers.

To prevent all subscriptions and publications being propagated throughout a cluster, leave this parameter blank on the system topics SYSTEM.BASE.TOPIC and SYSTEM.DEFAULT.TOPIC, except in special circumstances, for example, to support migration, documented elsewhere.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

'' The command is executed on the queue manager on which it was entered.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

COMMINFO(*comminfo-name***)**
The name of the Multicast communication information object associated with this topic object.

CUSTOM(*string***)**
The custom attribute for new features.

This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form `NAME(VALUE)`. Single quotes must be escaped with another single quote.

This description will be updated when features using this attribute are introduced. At the moment there are no possible values for *Custom*.

DEFPRTY(*integer***)**
The default priority of messages published to the topic.

(*integer***)**
The value must be in the range zero (the lowest priority), through to the MAXPRTY queue manager parameter (MAXPRTY is 9).

ASPARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DEFPSIST
Specifies the message persistence to be used when applications specify the MQPER_PERSISTENCE_AS_TOPIC_DEF option.

ASPARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

NO Messages on this queue are lost during a restart of the queue manager.

YES Messages on this queue survive a restart of the queue manager.

On z/OS, N and Y are accepted as synonyms of NO and YES.

DEFPRESP
Specifies the put response to be used when applications specify the MQPMO_RESPONSE_AS_DEF option.

ASPARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

SYNC Put operations to the queue that specify MQPMO_RESPONSE_AS_Q_DEF are issued as if MQPMO_SYNC_RESPONSE had been specified instead. Fields in the MQMD and MQPMO are returned by the queue manager to the application.

ASYNC
Put operations to the queue that specify MQPMO_RESPONSE_AS_Q_DEF are always issued as if MQPMO_ASYNC_RESPONSE had been specified instead. Some fields in the MQMD and MQPMO are not returned by the queue manager to the application; but an improvement in performance might be seen for messages put in a transaction and any non-persistent messages

DESCR(*string***)**
Plain-text comment. It provides descriptive information about the object when an operator issues the DISPLAY TOPIC command.

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

DURSUB
Specifies whether applications are permitted to make durable subscriptions on this topic.

ASPARENT
Whether durable subscriptions can be made on this topic is based on the setting of the closest parent administrative topic object in the topic tree.

NO Durable subscriptions cannot be made on this topic.

YES Durable subscriptions can be made on this topic.

LIKE(*topic-name***)**
The name of a topic. The topic parameters are used to model this definition.

If this field is not completed, and you do not complete the parameter fields related to the command, the values are taken from the default definition for topics on this queue manager.

Not completing this field is equivalent to specifying:

```
LIKE(SYSTEM.DEFAULT.TOPIC)
```

A default topic definition is provided, but it can be altered by the installation to the default values required. See Rules for naming IBM WebSphere MQ objects.

On z/OS, the queue manager searches page set zero for an object with the name you specify and a disposition of QMGR or COPY. The disposition of the LIKE object is not copied to the object you are defining.

Note:

1. QSGDISP (GROUP) objects are not searched.
2. LIKE is ignored if QSGDISP(COPY) is specified.

MCAST
Specifies whether multicast is allowable in the topic tree. The values are:

ASPARENT
The multicast attribute of the topic is inherited from the parent.

DISABLED
No multicast traffic is allowed at this node.

ENABLED
Multicast traffic is allowed at this node.

ONLY Only subscriptions from a multicast capable client are allowed.

MDURMDL(*string***)**
The name of the model queue to be used for durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming IBM WebSphere MQ objects). The maximum length is 48 characters.

If MDURMDL is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for MDURMDL.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.DURABLE

MNDURMDL(*string***)**
The name of the model queue to be used for non-durable subscriptions that request that the queue manager manages the destination of its publications (see Rules for naming IBM WebSphere MQ objects). The maximum length is 48 characters.

If MNDURMDL is blank, it operates in the same way as ASPARENT values on other attributes. The name of the model queue to be used is based on the closest parent administrative topic object in the topic tree with a value set for MNDURMDL.

The dynamic queue created from this model has a prefix of SYSTEM.MANAGED.NDURABLE.

NPMSGDLV

The delivery mechanism for non-persistent messages published to this topic:

ASPARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

ALL Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

ALLAVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ALLDUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT calls fails.

PMSGDLV

The delivery mechanism for persistent messages published to this topic:

ASPARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

ALL Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

ALLAVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ALLDUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT calls fails.

PROXYSUB

Controls when a proxy subscription is sent for this topic, or topic strings below this topic, to neighboring queue managers when in a publish/subscribe cluster or hierarchy. For more details, see More on routing mechanisms.

FIRSTUSE
For each unique topic string at or below this topic object, a proxy subscription is asynchronously sent to all neighboring queue managers in the following scenarios:

- When a local subscription is created.
- When a proxy subscription is received that must be propagated to further directly connected queue managers.

FORCE
A wildcard proxy subscription that matches all topic strings at and below this point in the topic tree is sent to neighboring queue managers even if no local subscriptions exist.

Note: The proxy subscription is sent when this value is set on DEFINE or ALTER. When set on a clustered topic, all queue managers in the cluster issue the wildcard proxy subscription to all other queue managers in the cluster.

PUB Controls whether messages can be published to this topic.

ASPARENT
Whether messages can be published to the topic is based on the setting of the closest parent administrative topic object in the topic tree.

ENABLED
Messages can be published to the topic (by suitably authorized applications).

DISABLED
Messages cannot be published to the topic.

PUBSCOPE
Determines whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster.

Note: You can restrict the behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message options.

ASPARENT
Determines whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster. This is based on the setting of the first parent administrative node found in the topic tree that relates to this topic.

QMGR
Publications for this topic are not propagated to connected queue managers.

ALL Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

QSGDISP
This parameter applies to z/OS only.

Specifies the disposition of the object within the group.

QSGDISP	DEFINE
COPY	The object is defined on the page set of the queue manager that executes the command using the QSGDISP(GROUP) object of the same name as the 'LIKE' object.

GROUP	The object definition resides in the shared repository but only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero:
	`DEFINE TOPIC(name)`
	`REPLACE QSGDISP(COPY)`
	
	The DEFINE for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

PRIVATE	Not permitted.
QMGR	The object is defined on the page set of the queue manager that executes the command.

REPLACE and NOREPLACE
Determines whether the existing definition (and on z/OS, with the same disposition) is to be replaced with this one. Any object with a different disposition is not changed.

REPLACE
If the object does exist, the effect is like issuing the ALTER command without the FORCE option and with *all* the other parameters specified.

(The difference between the ALTER command without the FORCE option, and the DEFINE command with the REPLACE option, is that ALTER does not change unspecified parameters, but DEFINE with REPLACE sets *all* the parameters. When you use REPLACE, unspecified parameters are taken either from the object named on the LIKE option, or from the default definition, and the parameters of the object being replaced, if one exists, are ignored.)

The command fails if both of the following are true:

- The command sets parameters that would require the use of the FORCE option if you were using the ALTER command.
- The object is open.

The ALTER command with the FORCE option succeeds in this situation.

NOREPLACE
The definition must not replace any existing definition of the object.

SUB Controls whether applications are to be permitted to subscribe to this topic.

ASPARENT
Whether applications can subscribe to the topic is based on the setting of the closest parent administrative topic object in the topic tree.

ENABLED
Subscriptions can be made to the topic (by suitably authorized applications).

DISABLED
Applications cannot subscribe to the topic.

SUBSCOPE
Determines whether this queue manager subscribes to publications in this queue manager or in the network of connected queue managers. If subscribing to all queue managers, the queue manager propagates subscriptions to them as part of a hierarchy or as part of a publish/subscribe cluster.

Note: You can restrict the behavior on a subscription-by-subscription basis, using **MQPMO_SCOPE_QMGR** on the Subscription Descriptor or **SUBSCOPE(QMGR)** on **DEFINE SUB**. Individual subscribers can override the **SUBSCOPE** setting of ALL by specifying the **MQSO_SCOPE_QMGR** subscription option when creating a subscription.

ASPARENT
Whether this queue manager subscribes to publications in the same way as the setting of the first parent administrative node found in the topic tree relating to this topic.

QMGR
Only publications that are published on this queue manager reach the subscriber.

ALL A publication made on this queue manager or on another queue manager reaches the subscriber. Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

TOPICSTR(*string***)**
The topic string represented by this topic object definition. This parameter is required and cannot contain the empty string.

The topic string must not be the same as any other topic string already represented by a topic object definition.

The maximum length of the string is 10,240 characters.

TYPE (topic-type)
If this parameter is used it must follow immediately after the *topic-name* parameter on all platforms except z/OS.

LOCAL
A local topic object.

USEDLQ
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue.

ASPARENT
Determines whether to use the dead-letter queue using the setting of the closest administrative topic object in the topic tree. This value is the default supplied with WebSphere MQ, but your installation might have changed it.

NO Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message. The MQPUT of an application to a topic fails in accordance with the settings of NPMSGDLV and PMSGDLV.

YES When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used. If the queue manager does not provide the name of a dead-letter queue, then the behavior is as for NO.

WILDCARD
The behavior of wildcard subscriptions with respect to this topic.

PASSTHRU
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic.

BLOCK
Subscriptions made to a wildcarded topic less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

The value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This scenario applies also if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the WILDCARD attribute is created using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

DELETE AUTHINFO:

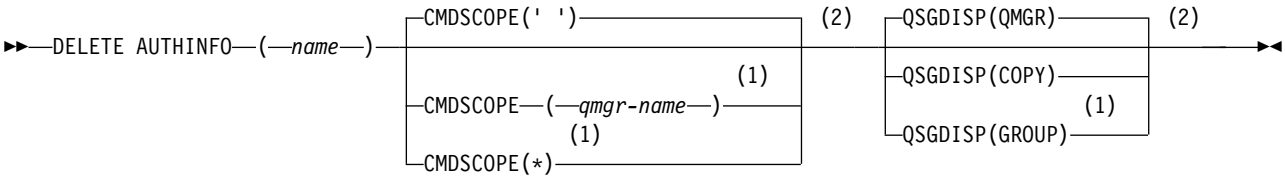
Use MQSC command DELETE AUTHINFO to delete an authentication information object.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- “Parameter descriptions for DELETE AUTHINFO” on page 561

Synonym: None

DELETE AUTHINFO



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on WebSphere MQ for z/OS.

2 Valid only on z/OS.

Parameter descriptions for DELETE AUTHINFO

(name) Name of the authentication information object. This is required.

The name must be that of an existing authentication information object.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP

The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE AUTHINFO(name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

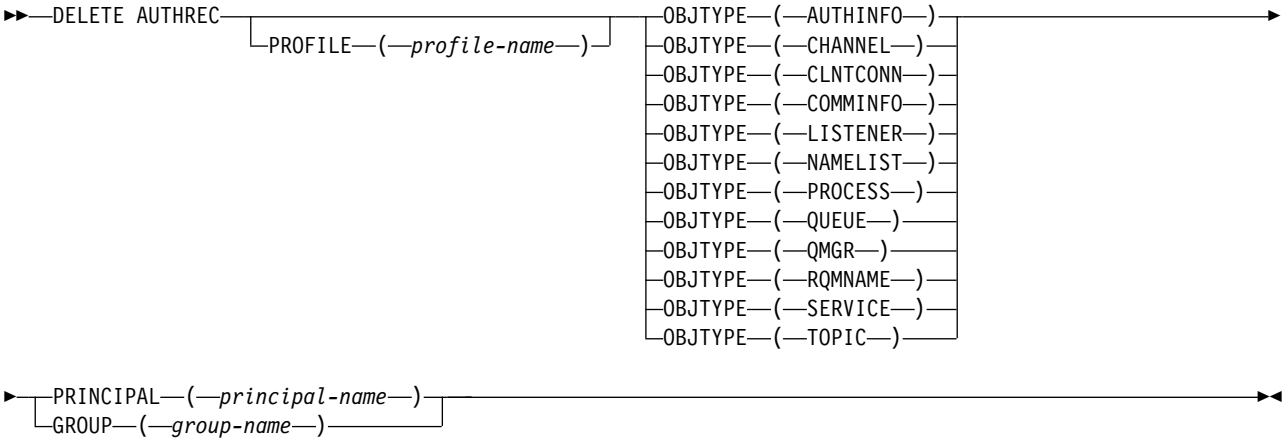
DELETE AUTHREC:

Use the MQSC command DELETE AUTHREC to delete authority records associated with a profile name.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions"

DELETE AUTHREC



Parameter descriptions

PROFILE(*profile-name*)
The name of the object or generic profile for which to remove the authority record. This parameter is required unless the **OBJTYPE** parameter is QMGR, in which case it can be omitted.

OBJTYPE
The type of object referred to by the profile. Specify one of the following values:

AUTHINFO
Authentication information record

CHANNEL
Channel

CLNTCONN
Client connection channel

COMMINFO
Communication information object

LISTENER
Listener

NAMELIST
Namelist

PROCESS
Process

QUEUE
Queue

QMGR
Queue manager

RQMNAME
Remote queue manager

SERVICE
Service

TOPIC
Topic

PRINCIPAL(*principal-name*)
A principal name. This is the name of a user for whom to remove authority records for the specified profile. On IBM WebSphere MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

You must specify either PRINCIPAL or GROUP.

GROUP(*group-name*)
A group name. This is the name of the user group for which to remove authority records for the specified profile. You can specify one name only and it must be the name of an existing user group.

For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

You must specify either PRINCIPAL or GROUP.

DELETE CHANNEL:

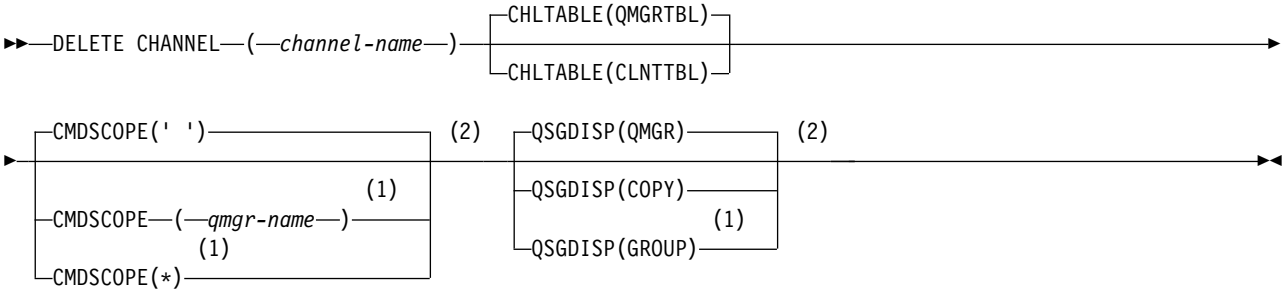
Use the MQSC command DELETE CHANNEL to delete a channel definition.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes" on page 564
- "Parameter descriptions" on page 564

Synonym: DELETE CHL

DELETE CHANNEL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

Notes for z/OS users:

1. The command fails if the channel initiator and command server have not been started, or the channel status is RUNNING, except client-connection channels, which can be deleted without the channel initiator or command server running.
2. You can only delete cluster-sender channels that have been created manually.

Parameter descriptions

(channel-name)
The name of the channel definition to be deleted. This is required. The name must be that of an existing channel.

CHLTABLE
Specifies the channel definition table that contains the channel to be deleted. This is optional.

QMGRTBL
The channel table is that associated with the target queue manager. This table does not contain any channels of type CLNTCONN. This is the default.

CLNTTBL
The channel table for CLNTCONN channels. On z/OS, this is associated with the target queue manager, but separate from the main channel table. On all other platforms, this channel table is normally associated with a queue manager, but can be a system-wide, queue manager independent channel table if you set up a number of environment variables. For more information about setting up environment variables, see Using WebSphere MQ environment variables.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

'' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP

The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE CHANNEL(channel-name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR

The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DELETE CHANNEL (MQTT):

Use the MQSC command DELETE CHANNEL to delete a IBM WebSphere MQ Telemetry channel definition.

IBM i	UNIX and Linux	Windows	z/OS
	✓	✓	

Note: For the telemetry server, AIX is the only supported UNIX platform.

The DELETE CHANNEL (MQTT) command is only valid for IBM WebSphere MQ Telemetry channels.

Synonym: DELETE CHL

DELETE CHANNEL

Parameter descriptions

(channel-name)
The name of the channel definition to be deleted. This is required. The name must be that of an existing channel.

CHLTYPE
This parameter is required. There is only one possible value: MQTT.

DELETE COMMINFO:

Use the MQSC command DELETE COMMINFO to delete a communication information object.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for DELETE COMMINFO"

Synonym: DEL COMMINFO

DELETE COMMINFO

Parameter descriptions for DELETE COMMINFO

(comminfo name)
The name of the communications information object to be deleted. This is required.

DELETE LISTENER:

Use the MQSC command DELETE LISTENER to delete a listener definition.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes for DELETE LISTENER" on page 567
- "Keyword and parameter descriptions for DELETE LISTENER" on page 567

Synonym: DELETE LSTR

DELETE LISTENER

Usage notes for DELETE LISTENER

1. The command fails if an application has the specified listener object open, or if the listener is currently running.

Keyword and parameter descriptions for DELETE LISTENER

(listener-name)

The name of the listener definition to be deleted. This is required. The name must be that of an existing listener defined on the local queue manager.

DELETE NAMELIST:

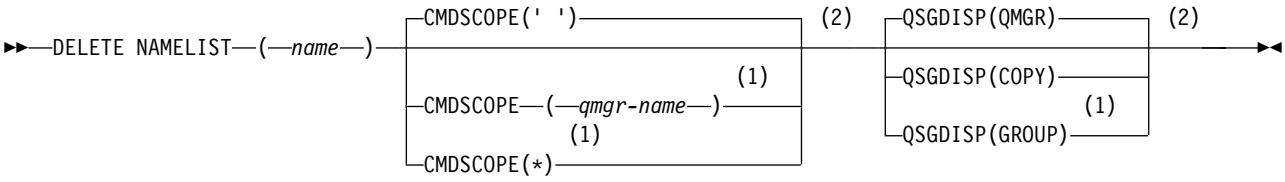
Use the MQSC command DELETE NAMELIST to delete a namelist definition.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for DELETE NAMELIST"

Synonym: DELETE NL

DELETE NAMELIST



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

On UNIX systems, the command is valid only on AIX, HP-UX, Linux, and Solaris.

Parameter descriptions for DELETE NAMELIST

You must specify which namelist definition you want to delete.

(name) The name of the namelist definition to be deleted. The name must be defined to the local queue manager.

If an application has this namelist open, the command fails.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP

The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE NAMELIST(name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR

The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DELETE PROCESS:

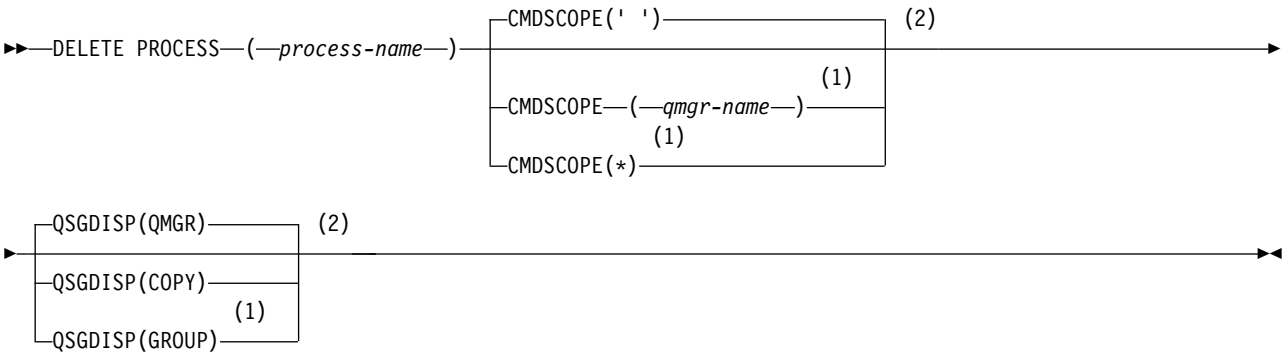
Use the MQSC command DELETE PROCESS to delete a process definition.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for DELETE PROCESS"

Synonym: DELETE PRO

DELETE PROCESS



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for DELETE PROCESS

You must specify which process definition you want to delete.

(process-name)
The name of the process definition to be deleted. The name must be defined to the local queue manager.

If an application has this process open, the command fails.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' '
The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP

This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP

The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE PROCESS(process-name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR

The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DELETE queues:

This section contains the following commands:

- "DELETE QALIAS" on page 572
- "DELETE QLOCAL" on page 573
- "DELETE QMODEL" on page 573
- "DELETE QREMOTE" on page 574

These commands are supported on the following platforms:

UNIX and Linux	Windows
✓	✓

Parameter descriptions for DELETE queues

(q-name)

The name of the queue must be defined to the local queue manager for all the queue types.

For an alias queue this is the local name of the alias queue to be deleted.

For a model queue this is the local name of the model queue to be deleted.

For a remote queue this is the local name of the remote queue to be deleted.

For a local queue this is the name of the local queue to be deleted. You must specify which queue you want to delete.

Note: A queue cannot be deleted if it contains uncommitted messages.

If an application has this queue open, or has open a queue that eventually resolves to this queue, the command fails. The command also fails if this queue is a transmission queue, and any queue that is, or resolves to, a remote queue that references this transmission queue, is open.

If this queue has a SCOPE attribute of CELL, the entry for the queue is also deleted from the cell directory.

AUTHREC

This parameter does not apply to z/OS.

Specifies whether the associated authority record is also deleted:

YES The authority record associated with the object is deleted. This is the default.

NO The authority record associated with the object is not deleted.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP or SHARED.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

PURGE and NOPURGE

Specifies whether any existing committed messages on the queue named by the DELETE command are to be purged for the delete command to work. The default is NOPURGE.

PURGE

The deletion is to go ahead even if there are committed messages on the named queue, and these messages are also to be purged.

NOPURGE

The deletion is not to go ahead if there are any committed messages on the named queue.

QSGDISP

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). If the object definition is shared, you do not need to delete it on every queue manager that is part of a queue-sharing group. (Queue-sharing groups are available only on WebSphere MQ for z/OS.)

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters

QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP
The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command, or any object defined using a command that had the parameters QSGDISP(SHARED), is not affected by this command.

If the deletion is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to make, or delete, local copies on page set zero:

```
DELETE queue(q-name) QSGDISP(COPY)
```

or, for a local queue only:

```
DELETE QLOCAL(q-name) NOPURGE QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

Note: You always get the NOPURGE option even if you specify PURGE. To delete messages on local copies of the queues, you must explicitly issue the command:

```
DELETE QLOCAL(q-name) QSGDISP(COPY) PURGE
```

for each copy.

QMGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

SHARED
This option applies only to local queues.

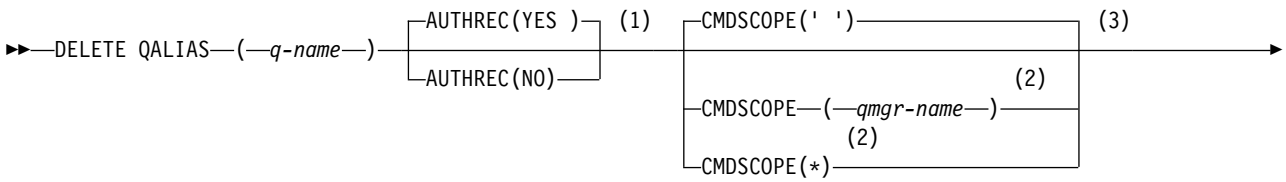
The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(SHARED). Any object residing on the page set of the queue manager that executes the command, or any object defined using a command that had the parameters QSGDISP(GROUP), is not affected by this command.

DELETE QALIAS:

Use DELETE QALIAS to delete an alias queue definition.

Synonym: DELETE QA

DELETE QALIAS



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

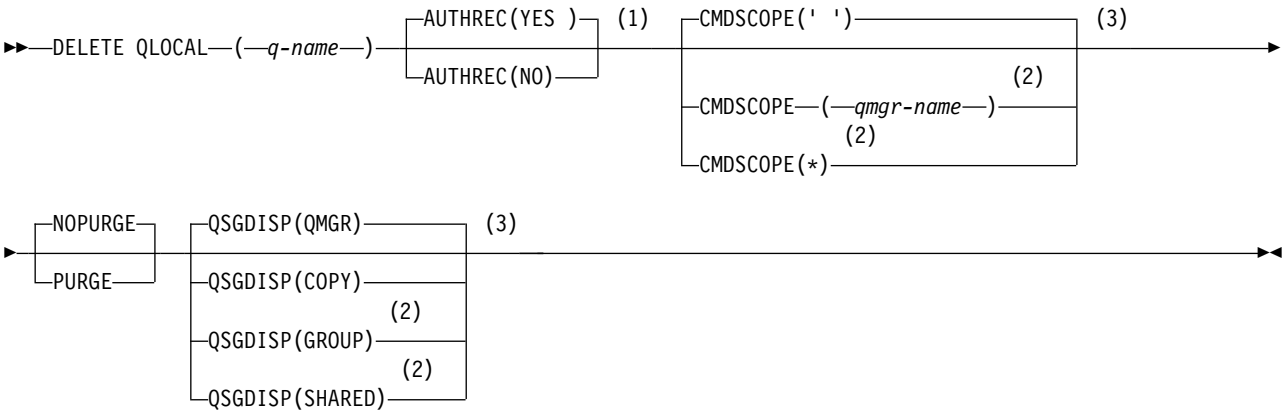
The parameters are described in "DELETE queues" on page 570.

DELETE QLOCAL:

Use DELETE QLOCAL to delete a local queue definition. You can specify that the queue must not be deleted if it contains messages, or that it can be deleted even if it contains messages.

Synonym: DELETE QL

DELETE QLOCAL



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

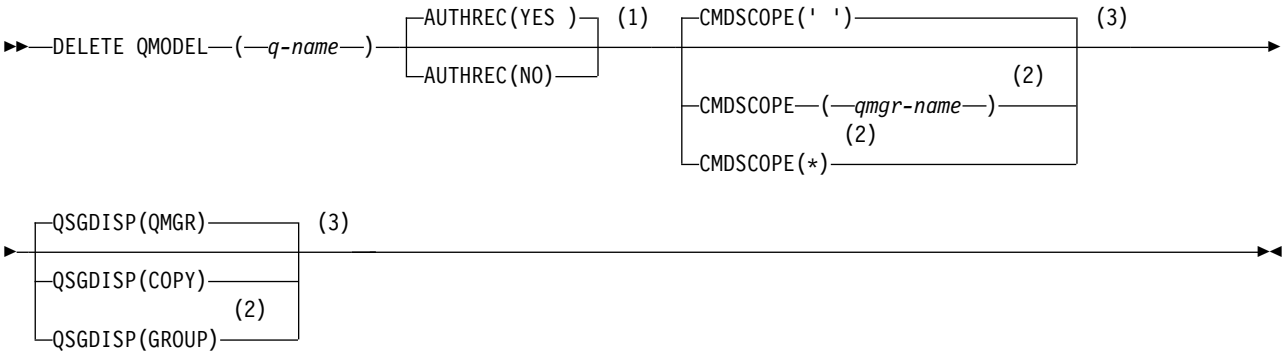
The parameters are described in "DELETE queues" on page 570.

DELETE QMODEL:

Use DELETE QMODEL to delete a model queue definition.

Synonym: DELETE QM

DELETE QMODEL



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

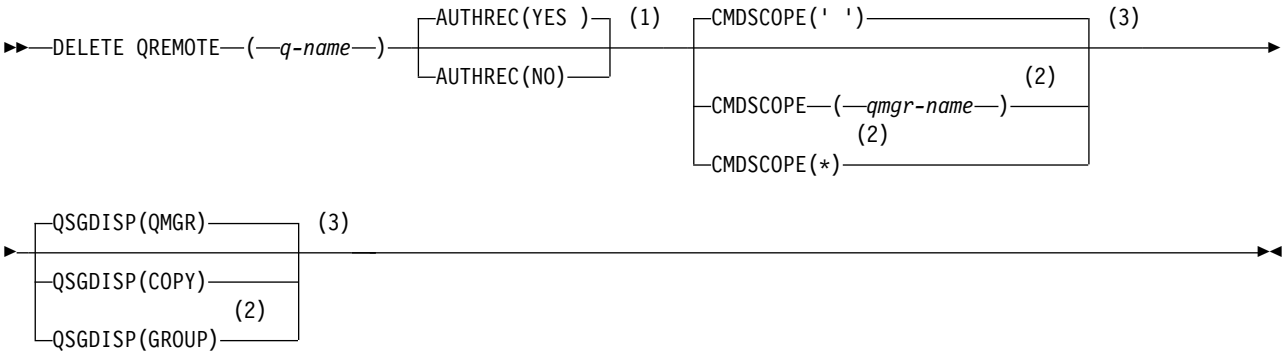
The parameters are described in "DELETE queues" on page 570.

DELETE QREMOTE:

Use DELETE QREMOTE to delete a local definition of a remote queue. It does not affect the definition of that queue on the remote system.

Synonym: DELETE QR

DELETE QREMOTE



Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

The parameters are described in "DELETE queues" on page 570.

DELETE SERVICE:

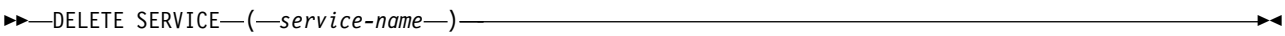
Use the MQSC command DELETE SERVICE to delete a service definition.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes for DELETE SERVICE"
- "Keyword and parameter descriptions for DELETE SERVICE"

Synonym:

DELETE SERVICE



Usage notes for DELETE SERVICE

1. The command fails if an application has the specified service object open, or if the service is currently running.

Keyword and parameter descriptions for DELETE SERVICE

(service-name)

The name of the service definition to be deleted. This is required. The name must be that of an existing service defined on the local queue manager.

DELETE SUB:

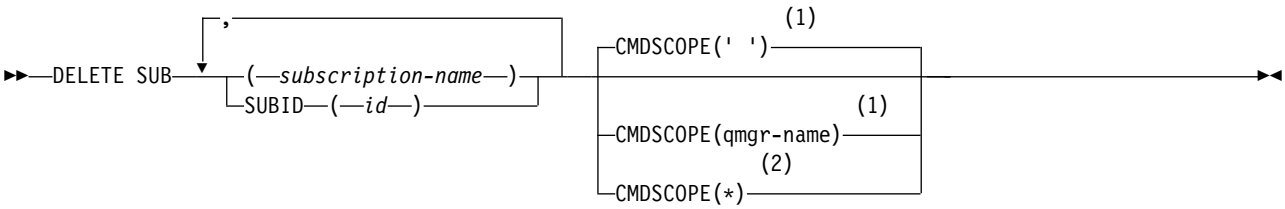
Use the MQSC command DELETE SUB to remove a durable subscription from the system. For a managed destination, any unprocessed messages left on the destination are removed.

IBM i	UNIX and Linux	Windows	z/OS
✓	✓	✓	CR

- Syntax diagram
- Usage Notes
- "Parameter descriptions for DELETE SUB" on page 576

Synonym: DEL SUB

DELETE SUB



Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes for DELETE SUB

You can specify either the name, the identifier, or both, of the subscription you want to delete.

Examples of valid forms:

```
DELETE SUB(xyz)
DELETE SUB SUBID(123)
DELETE SUB(xyz) SUBID(123)
```

Parameter descriptions for DELETE SUB

subscription-name
The local name of the subscription definition to be deleted.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is processed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is processed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

SUBID(*string***)**
The internal, unique key identifying a subscription.

DELETE TOPIC:

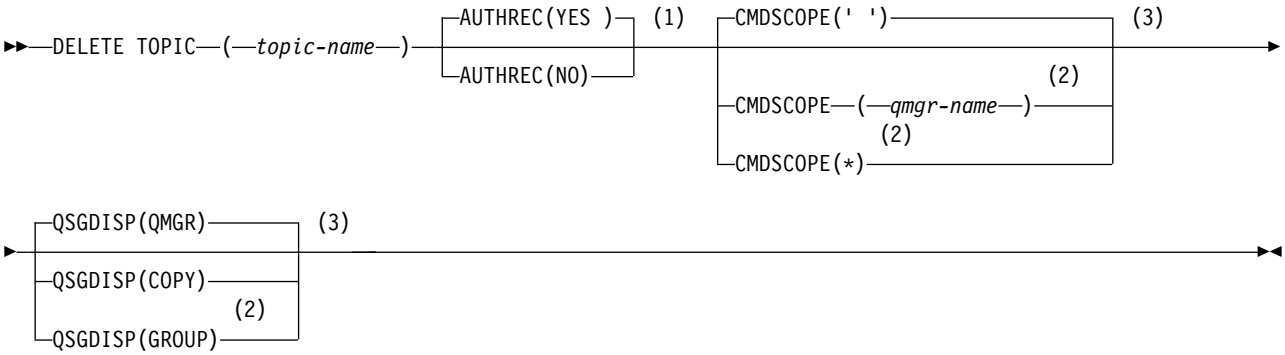
Use DELETE TOPIC to delete a WebSphere MQ administrative topic node.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for DELETE TOPIC" on page 577

Synonym: None

DELETE TOPIC



Notes:

1 Not valid on z/OS

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

Parameter descriptions for DELETE TOPIC

(topic-name)
The name of the administrative topic object to be deleted. This parameter is required.

The name must be that of an existing administrative topic object.

AUTHREC
This parameter does not apply to z/OS

Specifies whether the associated authority record is also deleted:

YES The authority record associated with the object is deleted. This is the default.

NO The authority record associated with the object is not deleted.

CMDSCOPE
This parameter applies to only z/OS and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP
This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves).

COPY The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(COPY). Any object residing in the shared repository, or any object defined using a command that had the parameters QSGDISP(QMGR), is not affected by this command.

GROUP
The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following command is generated and sent to all active queue managers in the queue-sharing group to make, or delete, local copies on page set zero:

```
DELETE TOPIC(topic-name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

QMGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameters QSGDISP(QMGR). Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

This is the default value.

DISPLAY AUTHINFO:

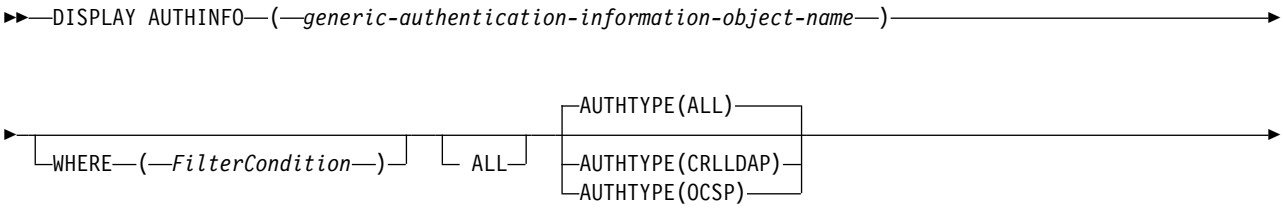
Use the MQSC command DISPLAY AUTHINFO to display the attributes of an authentication information object.

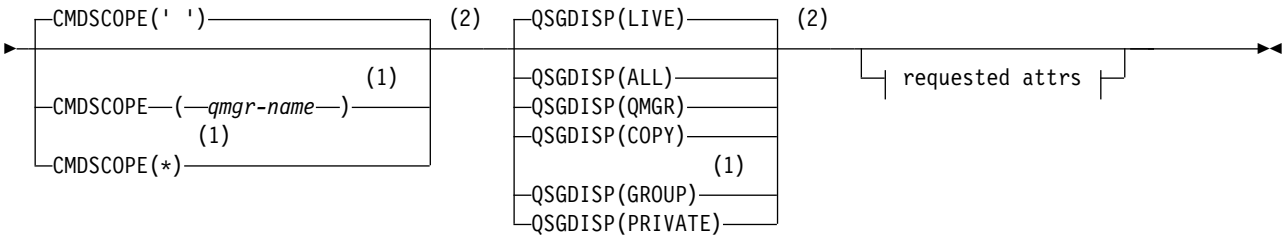
UNIX and Linux	Windows		
✓	✓	✓	2CR

- Syntax diagram
- "Parameter descriptions for DISPLAY AUTHINFO" on page 579
- "Requested parameters" on page 581

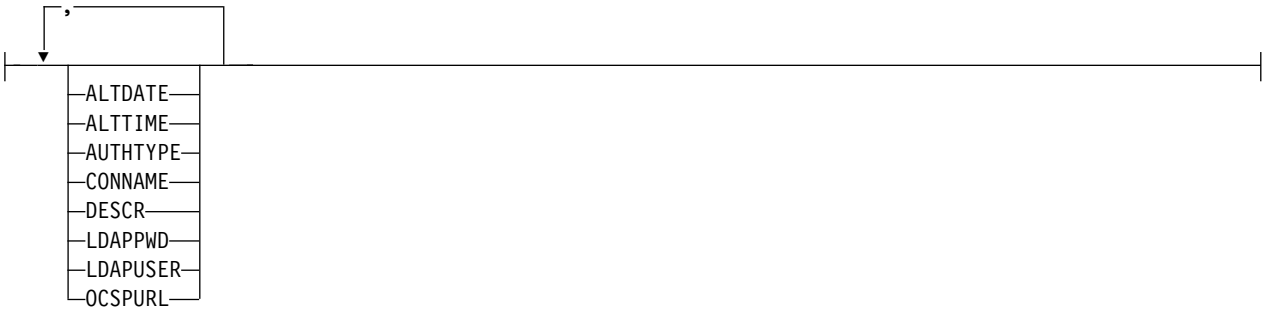
Synonym: DIS AUTHINFO

DISPLAY AUTHINFO





Requested attrs:



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group. You can use queue-sharing groups only on WebSphere MQ for z/OS.

2 Valid only on z/OS.

Parameter descriptions for DISPLAY AUTHINFO

(generic-authentication-information-object-name)
The name of the authentication information object to be displayed (see Rules for naming IBM WebSphere MQ objects). A trailing asterisk (*) matches all authentication information objects with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all authentication information objects.

WHERE
Specify a filter condition to display only those authentication information objects that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE or QSGDISP parameters as filter keywords.

operator
This is used to determine whether an authentication information object satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use any of the operators except LK and NL.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. You cannot use a generic filter-value with numeric values. Only a single trailing wildcard character (asterisk) is permitted.

  You can only use operators LK or NL for generic values on the DISPLAY AUTHINFO command.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic name and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

AUTHTYPE
Specifies the authentication information type of the objects for which information is to be displayed. Values are:

ALL
This is the default value and displays information for objects defined with AUTHTYPE(CRLLDAP) and with AUTHTYPE(OCSP).

CRLLDAP
Displays information only for objects defined with AUTHTYPE(CRLLDAP).

OCSP
Displays information only for objects defined with AUTHTYPE(OCSP).

QSGDISP
Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(LIVE) is specified or defaulted, or if QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

COPY Displays information only for objects defined with QSGDISP(COPY).

GROUP
Displays information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). Note that QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Displays information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the object names and their AUTHTYPEs, and, on z/OS, their QSGDISPs, are displayed.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`

AUTHTYPE
The type of the authentication information

CONNAME
The host name, IPv4 dotted decimal address, or IPv6 hexadecimal notation of the host on which the LDAP server is running. Applies only to objects with AUTHTYPE(CRLLDAP).

DESCR
Description of the authentication information object

LDAPPWD
Password associated with the Distinguished Name of the user on the LDAP server. If nonblank, this is displayed as asterisks (on all platforms except z/OS). Applies only to objects with AUTHTYPE(CRLLDAP).

LDAPUSER
Distinguished Name of the user on the LDAP server. Applies only to objects with AUTHTYPE(CRLLDAP).

OCSPURL
The URL of the OCSP responder used to check for certificate revocation. Applies only to objects with AUTHTYPE(OCSP).

See "Usage Notes for DEFINE AUTHINFO" on page 434 for more information about individual parameters.

DISPLAY AUTHREC:

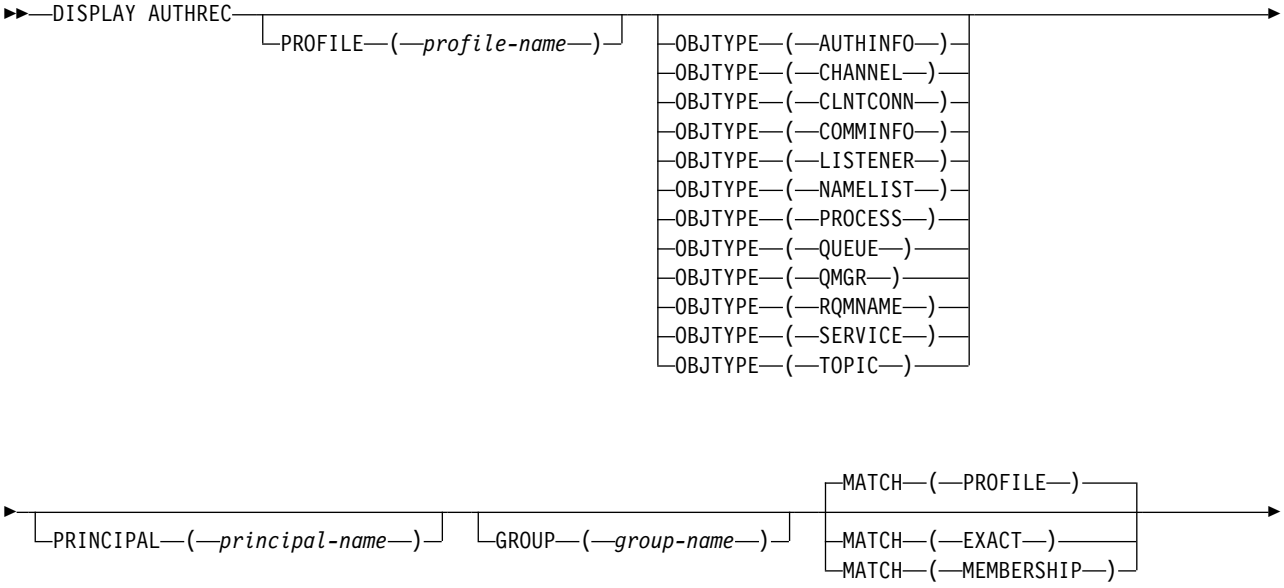
Use the MQSC command DISPLAY AUTHREC to display the authority records associated with a profile name.

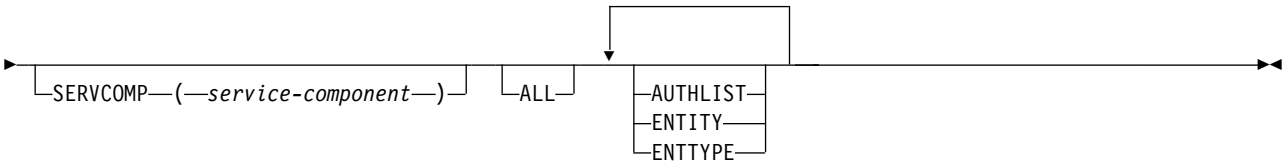
UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions" on page 583
- "Requested parameters" on page 584

Synonym: DIS AUTHREC

DISPLAY AUTHREC





Parameter descriptions

PROFILE(*profile-name*)
The name of the object or generic profile for which to display the authority records. If you omit this parameter, all authority records that satisfy the values of the other parameters are displayed.

OBJTYPE
The type of object referred to by the profile. Specify one of the following values:

AUTHINFO
Authentication information record

CHANNEL
Channel

CLNTCONN
Client connection channel

COMMINFO
Communication information object

LISTENER
Listener

NAMELIST
Namelist

PROCESS
Process

QUEUE
Queue

QMGR
Queue manager

RQMNAME
Remote queue manager

SERVICE
Service

TOPIC
Topic

If you omit this parameter, authority records for all object types are displayed.

PRINCIPAL(*principal-name*)
A principal name. This is the name of a user for whom to retrieve authorizations to the specified object. On IBM WebSphere MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

This parameter cannot be specified with GROUP.

GROUP(*group-name*)
A group name. This is the name of the user group on which to make the inquiry. You can specify one name only and it must be the name of an existing user group.

For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

This parameter cannot be specified with PRINCIPAL.

MATCH
Specify this parameter to control the set of authority records that is displayed. Specify one of the following values:

PROFILE
Return only those authority records which match the specified profile, principal, and group names. This means that a profile of ABCD results in the profiles ABCD, ABC*, and AB* being returned (if ABC* and AB* have been defined as profiles). If the profile name is a generic profile, only authority records which exactly match the specified profile name are returned. If a principal is specified, no profiles are returned for any group in which the principal is a member; only the profiles defined for the specified principal or group.

This is the default value.

MEMBERSHIP
Return only those authority records which match the specified profile, and the entity field of which matches the specified principal and the profiles pertaining to any groups in which the principal is a member that contribute to the cumulative authority for the specified entity.

If this option is specified, the PROFILE and OBJTYPE parameters must also be specified. In addition, either the PRINCIPAL or GROUP parameter must also be supplied. If OBJTYPE(QMGR) is specified, the profile name is optional.

EXACT
Return only those authority records which exactly match the specified profile name and EntityName. No matching generic profiles are returned unless the profile name is, itself, a generic profile. If a principal is specified, no profiles are returned for any group in which the principal is a member; only the profile defined for the specified principal or group.

SERVCOMP(*service-component*)
The name of the authorization service for which information is to be displayed.

If you specify this parameter, it specifies the name of the authorization service to which the authorizations apply. If you omit this parameter, the inquiry is made to the registered authorization services in turn in accordance with the rules for chaining authorization services.

ALL
Specify this parameter to display all of the authorization information available for the entity and the specified profile.

Requested parameters

You can request the following information about the authorizations:

AUTHLIST
Specify this parameter to display the list of authorizations.

ENTITY
Specify this parameter to display the entity name.

ENTTYPE
Specify this parameter to display the entity type.

DISPLAY AUTHSERV:

Use the MQSC command DISPLAY AUTHSERV to display information about the level of function supported by the installed authorization services.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions"
- "Requested parameters"

Synonym: DIS AUTHSERV

DISPLAY AUTHSERV

Parameter descriptions

ALL Specify this parameter to display all the information for each authorization service.

Requested parameters

You can request the following information for the authorization service:

IFVER
Specify this parameter to display the current interface version of the authorization service.

UIDSUPP
Specify this parameter to display whether the authorization service supports user IDs.

DISPLAY CHANNEL:

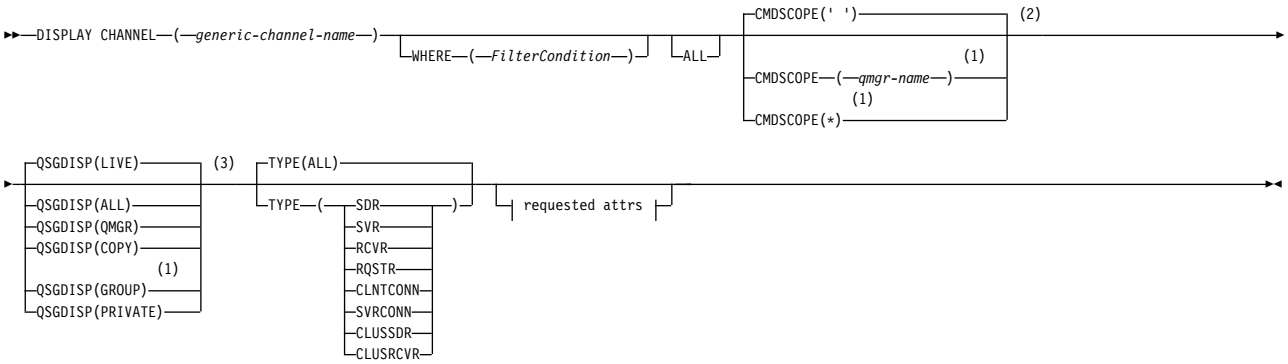
Use the MQSC command DISPLAY CHANNEL to display a channel definition.

UNIX and Linux	Windows
✓	✓

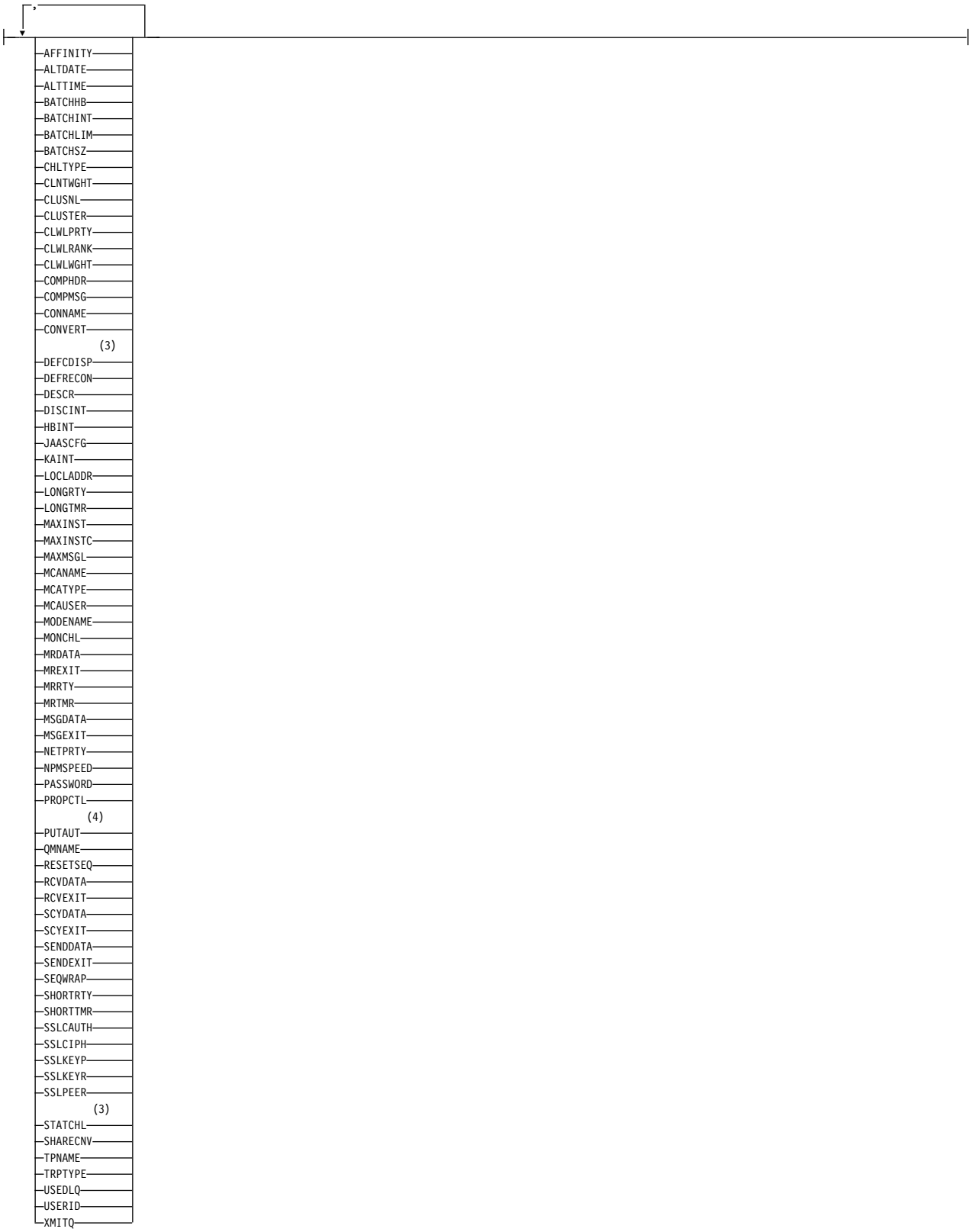
- Syntax diagram
- "Usage notes" on page 588
- "Parameter descriptions for DISPLAY CHANNEL" on page 588
- "Requested parameters" on page 591

Synonym: DIS CHL

DISPLAY CHANNEL



Requested attrs:



Notes:

1 Valid only on IBM WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

2 Not valid for z/OS client-connection channels.

3 Valid only on z/OS.

4 Valid only for RCVR, RQSTR, CLUSRCVR and (for z/OS only) SVRCONN channel types.

Usage notes

You can only display cluster-sender channels if they were created manually.

The values shown describe the current definition of the channel. If the channel has been altered since it was started, any currently running instance of the channel object might not have the same values as the current definition.

Parameter descriptions for DISPLAY CHANNEL

You must specify the name of the channel definition you want to display. It can be a specific channel name or a generic channel name. By using a generic channel name, you can display either:

- All channel definitions
- One or more channel definitions that match the specified name

(generic-channel-name)
The name of the channel definition to be displayed (see Rules for naming IBM WebSphere MQ objects). A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions.

WHERE
Specify a filter condition to display only those channels that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE, QSGDISP, or MCANAME parameters as filter keywords. You cannot use TYPE (or CHLTYPE) if it is also used to select channels. Channels of a type for which the filter keyword is not a valid attribute are not displayed.

operator
This is used to determine whether a channel satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the*filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the TYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

ALL Specify ALL to display the results of querying all the parameters. If ALL is specified, any request for a specific parameter is ignored. The result of querying with ALL is to return the results for all of the possible parameters.

This is the default, if you do not specify a generic name and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms, only requested attributes are displayed.

If no parameters are specified (and the ALL parameter is not specified or defaulted), the default is that the channel names only are displayed. On z/OS, the CHLTYPE and QSGDISP values are also displayed.

CMDSCOPE
This parameter specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP

Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

Note: In the QSGDISP(LIVE) case, this occurs only where a shared and a non-shared queue have the same name; such a situation should not occur in a well-managed system.

In a shared queue manager environment, use

```
DISPLAY CHANNEL(name) CMDSCOPE(*) QSGDISP(ALL)
```

to list ALL objects matching

```
name
```

in the queue-sharing group without duplicating those in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information only for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). Note that QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

TYPE This is optional. It can be used to restrict the display to channels of one type.

The value is one of the following:

ALL Channels of all types are displayed (this is the default).

SDR Sender channels only are displayed.

SVR Server channels only are displayed.

RCVR Receiver channels only are displayed.

RQSTR
Requester channels only are displayed.

CLNTCONN
Client-connection channels only are displayed.

SVRCONN
Server-connection channels only are displayed.

CLUSSDR
Cluster-sender channels only are displayed.).

CLUSRCVR
Cluster-receiver channels only are displayed.).

On all platforms, CHLTYPE(*type*) can be used as a synonym for this parameter.

Requested parameters

Specify one or more DISPLAY CHANNEL parameters that define the data to be displayed. You can specify the parameters in any order, but do not specify the same parameter more than once.

Some parameters are relevant only for channels of a particular type or types. Attributes that are not relevant for a particular type of channel cause no output, nor is an error raised. The following table shows the parameters that are relevant for each type of channel. There is a description of each parameter after the table. Parameters are optional unless the description states that they are required.

Table 81. Parameters that result in data being returned from the DISPLAY CHANNEL command

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
AFFINITY					✓			
ALTDATE	✓	✓	✓	✓	✓	✓	✓	✓
ALTTIME	✓	✓	✓	✓	✓	✓	✓	✓
AUTOSTART		✓	✓	✓		✓		
BATCHHB	✓	✓					✓	✓
BATCHINT	✓	✓		✓			✓	✓
BATCHLIM	✓	✓					✓	✓
BATCHSZ	✓	✓	✓	✓			✓	✓
channel-name	✓	✓	✓	✓	✓	✓	✓	✓
CHLTYPE	✓	✓	✓	✓	✓	✓	✓	✓
CLNTWGHT					✓			
CLUSNL							✓	✓
CLUSTER							✓	✓
CLWLPRTY							✓	✓

Table 81. Parameters that result in data being returned from the DISPLAY CHANNEL command (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
CLWLRANK							✓	✓
CLWLWGHT							✓	✓
COMPHDR	✓	✓	✓	✓	✓	✓	✓	✓
COMPMSG	✓	✓	✓	✓	✓	✓	✓	✓
CONNAME	✓	✓		✓	✓		✓	✓
CONVERT	✓	✓					✓	✓
DEFCDISP	✓	✓	✓	✓		✓		
DEFRECON					✓			
DESCR	✓	✓	✓	✓	✓	✓	✓	✓
DISCINT	✓	✓				✓	✓	✓
HBINT	✓	✓	✓	✓	✓	✓	✓	✓
KAINT	✓	✓	✓	✓	✓	✓	✓	✓
LOCLADDR	✓	✓		✓	✓		✓	✓
LONGRTY	✓	✓					✓	✓
LONGTMR	✓	✓					✓	✓
MAXINST						✓		
MAXINSTC						✓		
MAXMSGL	✓	✓	✓	✓	✓	✓	✓	✓
MCANAME	✓	✓		✓			✓	✓
MCATYPE	✓	✓		✓			✓	✓
MCAUSER	✓	✓	✓	✓		✓	✓	✓
MODENAME	✓	✓		✓	✓		✓	✓
MONCHL	✓	✓	✓	✓		✓	✓	✓
MRDATA			✓	✓				✓

Table 81. Parameters that result in data being returned from the DISPLAY CHANNEL command (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
MREXIT			✓	✓				✓
MRRTY			✓	✓				✓
MRTMR			✓	✓				✓
MSGDATA	✓	✓	✓	✓			✓	✓
MSGEXIT	✓	✓	✓	✓			✓	✓
NETPRTY								✓
NPMSPEED	✓	✓	✓	✓			✓	✓
PASSWORD	✓	✓		✓	✓		✓	
PROPCTL	✓	✓					✓	
PUTAUT			✓	✓		✓ [1]		✓
QMNAME					✓			
RESETSEQ	✓	✓	✓	✓			✓	✓
RCVDATA	✓	✓	✓	✓	✓	✓	✓	✓
RCVEXIT	✓	✓	✓	✓	✓	✓	✓	✓
SCYDATA	✓	✓	✓	✓	✓	✓	✓	✓
SCYEXIT	✓	✓	✓	✓	✓	✓	✓	✓
SENDDATA	✓	✓	✓	✓	✓	✓	✓	✓
SENDEXIT	✓	✓	✓	✓	✓	✓	✓	✓
SEQWRAP	✓	✓	✓	✓			✓	✓
SHARECNV						✓		
SHORTRTY	✓	✓					✓	✓
SHORTTMR	✓	✓					✓	✓
SSLCAUTH		✓	✓	✓		✓		✓
SSLCIPH	✓	✓	✓	✓	✓	✓	✓	✓

Table 81. Parameters that result in data being returned from the DISPLAY CHANNEL command (continued)

Parameter	SDR	SVR	RCVR	RQSTR	CLNT-CONN	SVR-CONN	CLUS-SDR	CLUS-RCVR
SSLPEER	✓	✓	✓	✓	✓	✓	✓	✓
STATCHL	✓	✓	✓	✓			✓	✓
TPNAME	✓	✓		✓	✓	✓	✓	✓
TRPTYPE	✓	✓	✓	✓	✓	✓	✓	✓
USEDLQ	✓	✓	✓	✓			✓	✓
USERID	✓	✓		✓	✓		✓	
XMITQ	✓	✓						
Note: 1. PUTAUT is valid for a channel type of SVRCONN on z/OS only.								

AFFINITY
The channel affinity attribute.

PREFERRED
Subsequent connections in a process attempt to use the same channel definition as the first connection.

NONE
All connections in a process select an applicable definition based on the weighting with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`.

AUTOSTART
Whether an LU 6.2 responder process should be started for the channel.

BATCHHB
The batch heartbeating value being used.

BATCHINT
Minimum batch duration.

BATCHLIM
Batch data limit.

The limit of the amount of data that can be sent through a channel.

BATCHSZ
Batch size.

CHLTYPE
Channel type.

The channel type is always displayed if you specify a generic channel name and do not request any other parameters. On z/OS, the channel type is always displayed.

On all platforms other than z/OS, TYPE can be used as a synonym for this parameter.

CLNTWGHT
The client channel weighting.

The special value 0 indicates that no random load balancing is performed and applicable definitions are selected in alphabetical order. If random load balancing is performed the value is in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest.

CLUSTER
The name of the cluster to which the channel belongs.

CLUSNL
The name of the namelist that specifies the list of clusters to which the channel belongs.

CLWLPRTY
The priority of the channel for the purposes of cluster workload distribution.

CLWLRANK
The rank of the channel for the purposes of cluster workload distribution.

CLWLWGHT
The weighting of the channel for the purposes of cluster workload distribution.

COMPHDR
The list of header data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

COMPMSG
The list of message data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

CONNAME
Connection name.

CONVERT
Whether sender should convert application message data.

DEFCDISP
Specifies the default channel disposition of the channels for which information is to be returned. If this keyword is not present, channels of all default channel dispositions are eligible.

ALL Channels of all default channel dispositions are displayed.

This is the default setting.

PRIVATE
Only channels where the default channel disposition is PRIVATE are displayed.

SHARED
Only channels where the default channel disposition is FIXSHARED or SHARED are displayed.

Note: This does not apply to client-connection channel types on z/OS.

DESCR
Default client reconnection option.

DESCR
Description.

DISCINT
Disconnection interval.

HBINT
Heartbeat interval.

KAINT
KeepAlive timing for the channel.

LOCLADDR
Local communications address for the channel.

LONGRTY
Long retry count.

LONGTMR
Long retry timer.

MAXINST(*integer***)**
The maximum number of instances of a server-connection channel that are permitted to run simultaneously.

MAXINSTC(*integer***)**
The maximum number of instances of a server-connection channel, started from a single client, that are permitted to run simultaneously.

Note: In this context, connections originating from the same remote network address are regarded as coming from the same client.

MAXMSGL
Maximum message length for channel.

MCANAME
Message channel agent name.

You cannot use MCANAME as a filter keyword.

MCATYPE
Whether message channel agent runs as a separate process or a separate thread.

MCAUSER
Message channel agent user identifier.

MODENAME
LU 6.2 mode name.

MONCHL
Online monitoring data collection.

MRDATA
Channel message-retry exit user data.

MREXIT
Channel message-retry exit name.

MRRTY
Channel message-retry count.

MRTMR
Channel message-retry time.

MSGDATA
Channel message exit user data.

MSGEXIT
Channel message exit names.

NETPRTY
The priority for the network connection.

NPMSPEED
Nonpersistent message speed.

PASSWORD
Password for initiating LU 6.2 session (if nonblank, this is displayed as asterisks on all platforms except z/OS).

PROPCTL
Message property control.

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

This parameter is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

This parameter is optional.

Permitted values are:

COMPAT
This is the default value.

Message properties	Result
The message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All optional message properties (where the **Support** value is MQPD_SUPPORT_OPTIONAL), except those in the message descriptor or extension, are placed in one or more MQRFH2 headers in the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All message properties, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL	The message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.
The message contains one or more properties where the **Support** field of the property descriptor is set to MQPD_SUPPORT_OPTIONAL but other fields of the property descriptor are set to non-default values	The properties with non-default values are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the *content='properties'* attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a V6 or prior queue manager.

NONE
All properties of the message, except those in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

If the message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL then the message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.

ALL All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

PUTAUT
Put authority.

QMNAME
Queue manager name.

RESETSEQ
Pending reset sequence number.

This is the sequence number from an outstanding request and it indicates a user RESET CHANNEL command request is outstanding.

A value of zero indicates that there is no outstanding RESET CHANNEL. The value can be in the range 1 - 999999999.

This parameter is not applicable on z/OS.

RCVDATA
Channel receive exit user data.

RCVEXIT
Channel receive exit names.

SCYDATA
Channel security exit user data.

SCYEXIT
Channel security exit names.

SENDDATA
Channel send exit user data.

SENDEXIT
Channel send exit names.

SEQWRAP
Sequence number wrap value.

SHARECNV
Sharing conversations value.

SHORTRTY
Specifies the maximum number of times that the channel is to try allocating a session to its partner.

SHORTTMR
Short retry timer.

SSLCAUTH
Whether SSL client authentication is required.

SSLCIPH
Cipher specification for the SSL connection.

SSLPEER
Filter for the Distinguished Name from the certificate of the peer queue manager or client at the other end of the channel.

STATCHL
Statistics data collection.

TPNAME
LU 6.2 transaction program name.

TRPTYPE
Transport type.

USEDLQ
Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

USERID
User identifier for initiating LU 6.2 session.

XMITQ
Transmission queue name.

For more details of these parameters, see "DEFINE CHANNEL" on page 437.

DISPLAY CHANNEL (MQTT):

Use the MQSC command DISPLAY CHANNEL to display a IBM WebSphere MQ Telemetry channel definition.

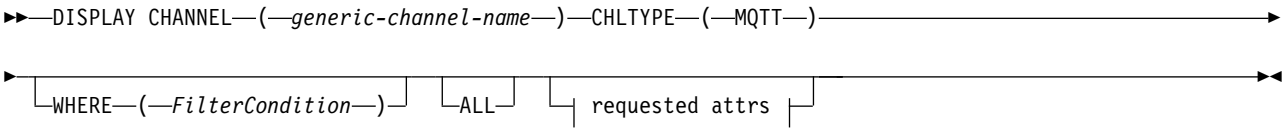
IBM i	UNIX and Linux	Windows	z/OS
	✓	✓	

Note: For the telemetry server, AIX is the only supported UNIX platform.

- Syntax diagram
- "Parameter descriptions for DISPLAY CHANNEL"
- "Requested parameters" on page 601

Synonym: DIS CHL

DISPLAY CHANNEL



Requested attrs:

DISPLAY CHANNEL (MQTT) command is only valid for WebSphere MQ Telemetry channels.

Parameter descriptions for DISPLAY CHANNEL

You must specify the name of the channel definition you want to display. This can be a specific channel name or a generic channel name. By using a generic channel name, you can display either:

- All channel definitions

- One or more channel definitions that match the specified name

(generic-channel-name)
The name of the channel definition to be displayed (see Rules for naming IBM WebSphere MQ objects). A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions.

WHERE
Specify a filter condition to display only those channels that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE, QSGDISP, or MCANAME parameters as filter keywords. You cannot use TYPE (or CHLTYPE) if it is also used to select channels. Channels of a type for which the filter keyword is not a valid attribute are not displayed.

operator
This is used to determine whether a channel satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the*filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:
- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the TYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

ALL Specify ALL to display the results of querying all the parameters. If ALL is specified, any request for a specific parameter is ignored. The result of querying with ALL is to return the results for all of the possible parameters.

This is the default, if you do not specify a generic name and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms, only requested attributes are displayed.

If no parameters are specified (and the ALL parameter is not specified or defaulted), the default is that the channel names only are displayed. On z/OS, the CHLTYPE and QSGDISP values are also displayed.

TYPE This is optional. It can be used to restrict the display to channels of one type.

The value is one of the following:

MQTT
Telemetry channels only are displayed.

CHLTYPE(*type*) can be used as a synonym for this parameter.

Requested parameters

Specify one or more DISPLAY CHANNEL parameters that define the data to be displayed. You can specify the parameters in any order, but do not specify the same parameter more than once.

Some parameters are relevant only for channels of a particular type or types. Attributes that are not relevant for a particular type of channel cause no output, nor is an error raised. The following table shows the parameters that are relevant for each type of channel. There is a description of each parameter after the table. Parameters are optional unless the description states that they are required.

BACKLOG
The number of outstanding connection requests that the telemetry channel can support at any one time. When the backlog limit is reached, any further clients trying to connect will be refused connection until the current backlog is processed. The value is in the range 0 - 999999999. The default value is 4096.

CHLTYPE
Channel type.

There is only one valid value for this parameter: MQTT.

JAASCFG
The file path of the JAAS configuration.

LOCLADDR
Local communications address for the channel.

MCAUSER
Message channel agent user identifier.

PORT The port number on which the telemetry (MQXR) service listens for TCP/IP connections to this channel.

SSLCAUTH
Whether SSL client authentication is required.

SSLCIPH
When SSLCIPH is used with a telemetry channel, it means "SSL Cipher Suite".

SSLKEYP
The store for digital certificates and their associated private keys. If you do not specify a key file, SSL is not used.

SSLKEYR
The password for the key repository. If no passphrase is entered, then unencrypted connections must be used.

USECLTID
Decide whether you want to use the MQTT client ID for the new connection as the IBM WebSphere MQ user ID for that connection. If this property is specified, the user name supplied by the client is ignored.

For more details of these parameters, see "DEFINE CHANNEL (MQTT)" on page 489.

DISPLAY CHLAUTH:

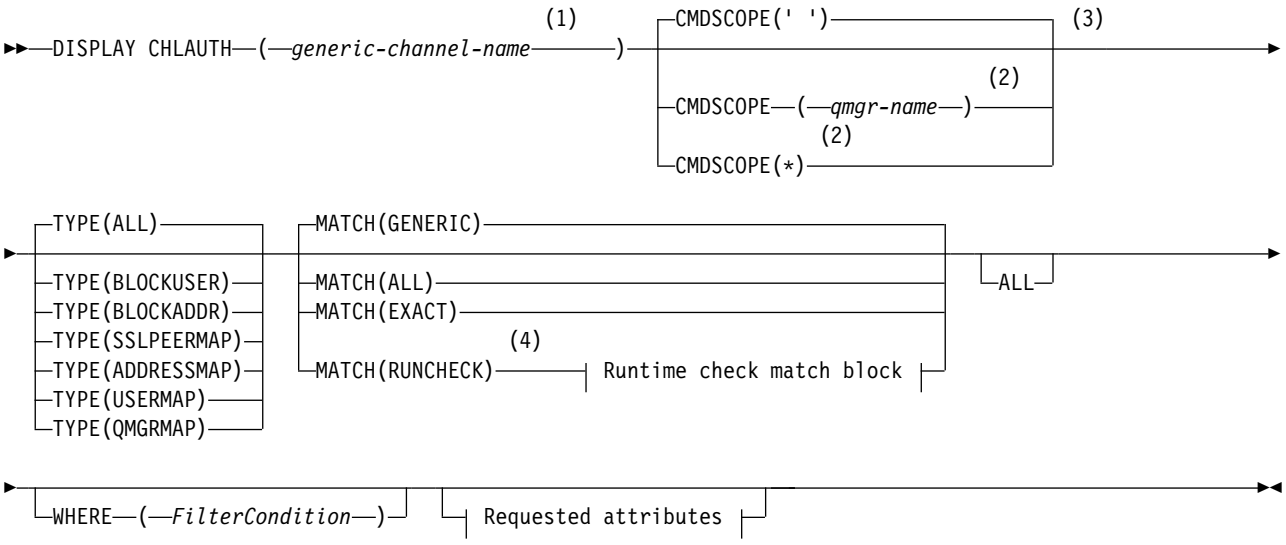
Use the MQSC command DISPLAY CHLAUTH to display the attributes of a channel authentication record.

UNIX and Linux	Windows
✓	✓

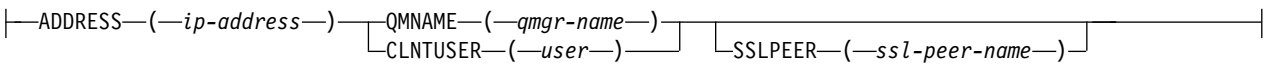
- Syntax diagram
- Parameters

Synonym: DIS CHLAUTH

DISPLAY CHLAUTH



Runtime check match block:



Requested attributes:

Notes:

1 Must be * with TYPE(BLOCKADDR) and cannot be generic with MATCH(RUNCHECK)

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Must be combined with TYPE(ALL)

Parameters

generic-channel-name
The name of the channel or set of channels to display. You can use the asterisk (*) as a wildcard to specify a set of channels. When **MATCH** is RUNCHECK this parameter must not be generic.

ADDRESS
The IP address to be matched.

This parameter is valid only when **MATCH** is RUNCHECK and must not be generic.

ALL
Specify this parameter to display all attributes. If this keyword is specified, any attributes that are requested specifically have no effect; all attributes are still displayed.

This is the default behavior if you do not specify a generic name and do not request any specific attributes.

CLNTUSER
The client user ID to be matched.

This parameter is valid only when **MATCH** is RUNCHECK and must not be generic.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is run when the queue manager is a member of a queue-sharing group.

'' The command is run on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is run on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is run on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

CUSTOM
Reserved for future use.

MATCH
Indicates the type of matching to be applied.

RUNCHECK
Returns the record that will be matched by a specific inbound channel at run time if it connects into this queue manager. The specific inbound channel is described by providing values that are not generic for:

- the channel name
- ADDRESS attribute
- SSLPEER attribute, only if the inbound channel will use SSL or TLS
- QMNAME or CLNTUSER attribute, depending on whether the inbound channel will be a client or queue manager channel

If the record discovered has WARN set to YES, a second record might also be displayed to show the actual record the channel will use at runtime. This parameter must be combined with TYPE(ALL).

EXACT
Return only those records which exactly match the channel profile name supplied. If there are no asterisks in the channel profile name, this option returns the same output as MATCH(GENERIC).

GENERIC
Any asterisks in the channel profile name are treated as wild cards. If there are no asterisks in the channel profile name, this returns the same output as MATCH(EXACT). For example, a profile of ABC* could result in records for ABC, ABC*, and ABCD being returned.

ALL Return all possible records that match the channel profile name supplied. If the channel name is generic in this case, all records that match the channel name are returned even if more specific matches exist. For example, a profile of SYSTEM.*.SVRCONN could result in records for SYSTEM.*, SYSTEM.DEF.*, SYSTEM.DEF.SVRCONN, and SYSTEM.ADMIN.SVRCONN being returned.

QMNAME
The name of the remote partner queue manager to be matched

This parameter is valid only when **MATCH** is RUNCHECK and must not be generic.

SSLPEER

The Subject Distinguished Name of the certificate to be matched.

The **SSLPEER** value is specified in the standard form used to specify a Distinguished Name.

This parameter is valid only when **MATCH** is RUNCHECK and must not be generic.

TYPE
The type of Channel Authentication Record for which to display details. Possible values are:

- ALL
- BLOCKUSER
- BLOCKADDR
- SSLPEERMAP
- ADDRESSMAP
- USERMAP
- QMGRMAP

WHERE

Specify a filter condition to display only those channel authentication records that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword

Any parameter that can be used to display attributes for this DISPLAY command.

operator

This is used to determine whether a channel authentication record satisfies the filter value on the given filter keyword. The operators are as follows:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

filter-value

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, the value can be either explicit or generic:

- An explicit value, that is a valid value for the attribute being tested.

  You can use any of the operators except LK and NL. However, if the value is one from a possible set of values returnable on a parameter (for example, the value ALL on the MATCH parameter), you can only use EQ or NE.
- A generic value. This is a character string with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

You can only use operators LK or NL for generic values.

- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

TYPE
The type of channel authentication record

SSLPEER
The Distinguished Name of the certificate.

ADDRESS
The IP address

CLNTUSER
The client asserted user ID

QMNAME
The name of the remote partner queue manager

MCAUSER
The user identifier to be used when the inbound connection matches the SSL DN, IP address, client asserted user ID or remote queue manager name supplied.

ADDRLIST
A list of IP address patterns which are banned from connecting into this queue manager on any channel.

USERLIST
A list of user IDs which are banned from use of this channel or set of channels.

ALTDATE
The date on which the channel authentication record was last altered, in the format *yyyy-mm-dd*.

ALTTIME
The time on which the channel authentication record was last altered, in the form *hh.mm.ss*.

DESCR
Descriptive information about the channel authentication record.

CUSTOM
Reserved for future use.

Related information:
Channel authentication records

Generic IP addresses:

In the various commands that create and display channel authentication records, you can specify certain parameters as either a single IP address or a pattern to match a set of IP addresses.

When you create a channel authentication record, using the MQSC command SET CHLAUTH or the PCF command Set Channel Authentication Record , you can specify a generic IP address in various contexts.

You can also specify a generic IP address in the filter condition when you display a channel authentication record using the commands DISPLAY CHLAUTH or Inquire Channel Authentication Records .

You can specify the address in any of the following ways:

- a single IPv4 address, such as 192.0.2.0
- a pattern based on an IPv4 address, including an asterisk (*) as a wildcard. The wildcard represents one or more parts of the address, depending on context. For example, the following are all valid values:
  - 192.0.2.*
  - 192.0.*
  - 192.0.*.2
  - 192.*.2
  - *
- a pattern based on an IPv4 address, including a hyphen (-) to indicate a range, for example 192.0.2.1-8
- a pattern based on an IPv4 address, including both an asterisk and a hyphen, for example 192.0.*.1-8
- a single IPv6 address, such as 2001:DB8:0:0:0:0:0:0
- a pattern based on an IPv6 address including an asterisk (*) as a wildcard. The wildcard represents one or more parts of the address, depending on context. For example, the following are all valid values:
  - 2001:DB8:0:0:0:0:0:*
  - 2001:DB8:0:0:0:*
  - 2001:DB8:0:0:0:*:0:1
  - 2001:*:1
  - *
- a pattern based on an IPv6 address, including a hyphen (-) to indicate a range, for example 2001:DB8:0:0:0:0:0:0-8
- a pattern based on an IPv6 address, including both an asterisk and a hyphen, for example 2001:DB8:0:0:0:*:0:0-8

If your system supports both IPv4 and IPv6, you can use either address format. IBM WebSphere MQ recognizes IPv4 mapped addresses in IPv6.

Certain patterns are invalid:

- A pattern cannot have fewer than the required number of parts, unless the pattern ends with a single trailing asterisk. For example 192.0.2 is invalid, but 192.0.2.* is valid.
- A trailing asterisk must be separated from the rest of the address by the appropriate part separator (a dot (.) for IPv4, a colon (:) for IPv6). For example, 192.0* is not valid because the asterisk is not in a part of its own.
- A pattern may contain additional asterisks provided that no asterisk is adjacent to the trailing asterisk. For example, 192.*.2.* is valid, but 192.0.*.* is not valid.
- An IPv6 address pattern cannot contain a double colon and a trailing asterisk, because the resulting address would be ambiguous. For example, 2001::* could expand to 2001:0000:*, 2001:0000:0000:* and so on

Related information:
Mapping an IP address to an MCAUSER user ID

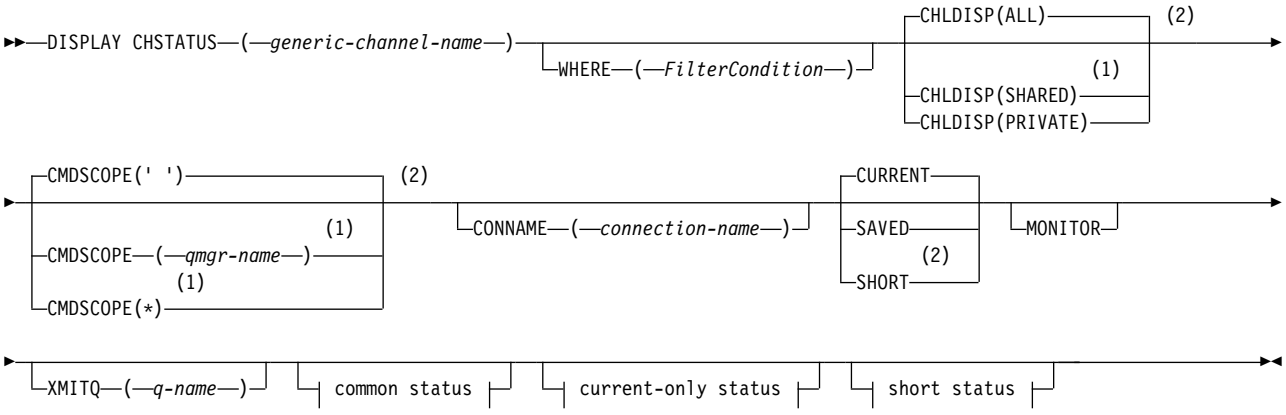
DISPLAY CHSTATUS:

Use the MQSC command DISPLAY CHSTATUS to display the status of one or more channels.

UNIX and Linux	Windows
✓	✓

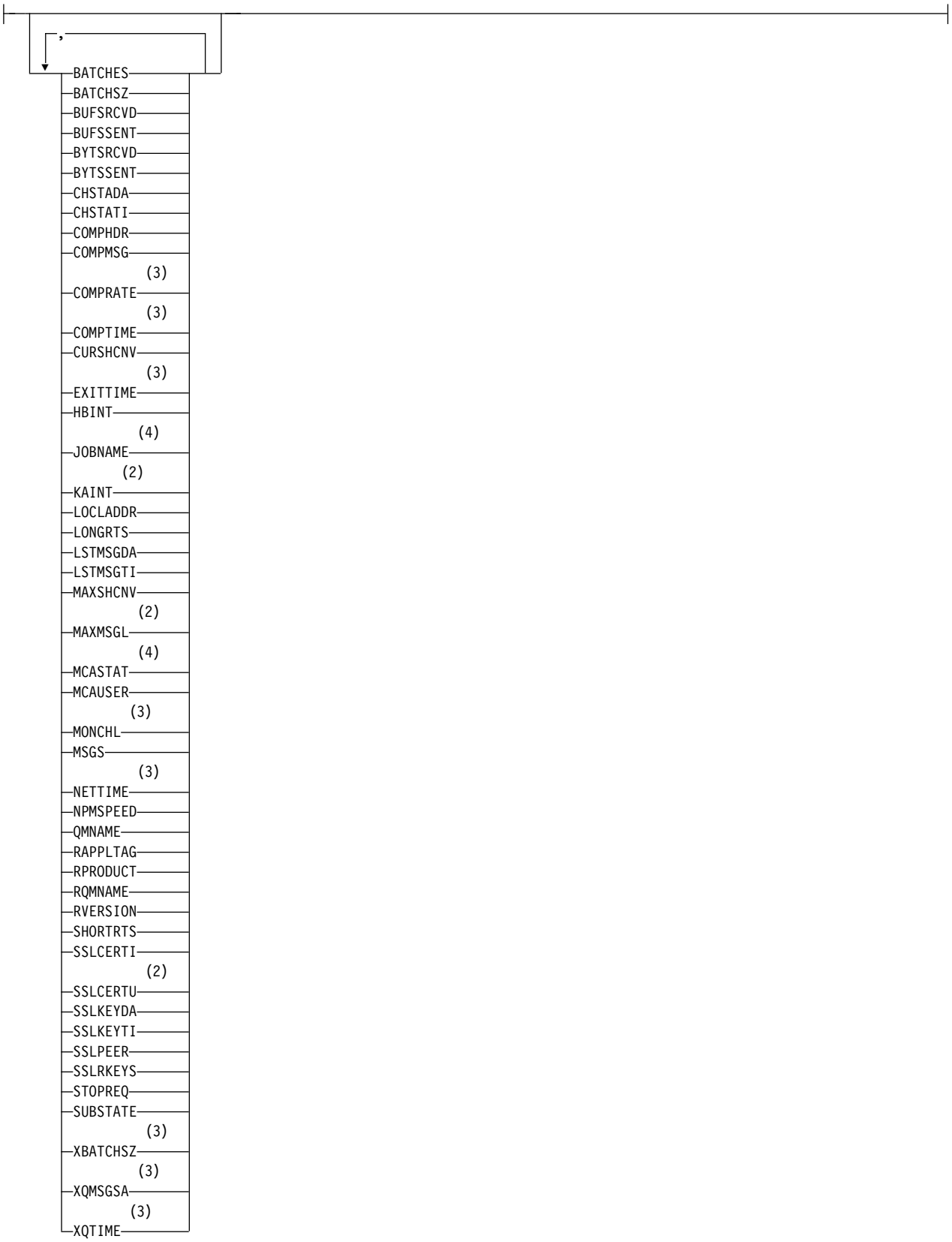
Synonym: DIS CHS

DISPLAY CHSTATUS

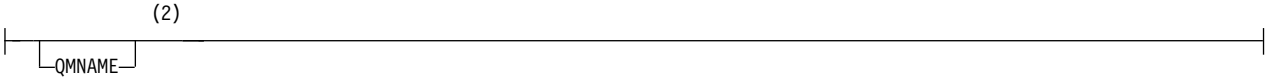


Common status:

Current-only status:



Short status:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Also displayed by selection of the MONITOR parameter.

4 Ignored if specified on z/OS.

Usage notes for DISPLAY CHSTATUS

On z/OS:

1. The command fails if the channel initiator has not been started.
2. The command server must be running.
3. On z/OS, if any numeric parameter exceeds 999,999,999, it is displayed as 999999999.
4. The status information that is returned for various combinations of CHLDISP, CMDSCOPE, and status type are summarized in Table 82, Table 83, and Table 84 on page 611.

Table 82. CHLDISP and CMDSCOPE for DISPLAY CHSTATUS CURRENT

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	Common and current-only status for current private channels on the local queue manager	Common and current-only status for current private channels on the named queue manager	Common and current-only status for current private channels on all queue managers
SHARED	Common and current-only status for current shared channels on the local queue manager	Common and current-only status for current shared channels on the named queue manager	Common and current-only status for current shared channels on all queue managers
ALL	Common and current-only status for current private and shared channels on the local queue manager	Common and current-only status for current private and shared channels on the named queue manager	Common and current-only status for current private and shared channels on all active queue managers

Table 83. CHLDISP and CMDSCOPE for DISPLAY CHSTATUS SHORT

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	STATUS and short status for current private channels on the local queue manager	STATUS and short status for current private channels on the named queue manager	STATUS and short status for current private channels on all active queue managers
SHARED	STATUS and short status for current shared channels on all active queue managers in the queue-sharing group	Not permitted	Not permitted

Table 83. CHLDISP and CMDSCOPE for DISPLAY CHSTATUS SHORT (continued)

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
ALL	STATUS and short status for current private channels on the local queue manager and current shared channels in the queue-sharing group (1)	STATUS and short status for current private channels on the named queue manager	STATUS and short status for current private, and shared, channels on all active queue managers in the queue-sharing group (1)
Note: 1. In this case you get two separate sets of responses to the command on the queue manager where it was entered; one for PRIVATE and one for SHARED.			

Table 84. CHLDISP and CMDSCOPE for DISPLAY CHSTATUS SAVED

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	Common status for saved private channels on the local queue manager	Common status for saved private channels on the named queue manager	Common status for saved private channels on all active queue managers
SHARED	Common status for saved shared channels on all active queue managers in the queue-sharing group	Not permitted	Not permitted
ALL	Common status for saved private channels on the local queue manager and saved shared channels in the queue-sharing group	Common status for saved private channels on the named queue manager	Common status for saved private, and shared, channels on all active queue managers in the queue-sharing group

Parameter descriptions for DISPLAY CHSTATUS

You must specify the name of the channel for which you want to display status information. This can be a specific channel name or a generic channel name. By using a generic channel name, you can display either the status information for all channels, or status information for one or more channels that match the specified name.

You can also specify whether you want the current status data (of current channels only), or the saved status data of all channels.

Status for all channels that meet the selection criteria is displayed, whether the channels were defined manually or automatically.

There are three classes of data available for channel status. These are **saved**, **current**, and (on z/OS only) **short**.

The status fields available for saved data are a subset of the fields available for current data and are called **common** status fields. Note that although the common data *fields* are the same, the data *values* might be different for saved and current status. The rest of the fields available for current data are called **current-only** status fields.

- **Saved** data consists of the common status fields noted in the syntax diagram.
  - For a sending channel data is updated before requesting confirmation that a batch of messages has been received and when confirmation has been received

  - For a receiving channel data is reset just before confirming that a batch of messages has been received
  - For a server connection channel no data is saved.
  - Therefore, a channel that has never been current cannot have any saved status.

    Note: Status is not saved until a persistent message is transmitted across a channel, or a nonpersistent message is transmitted with a NPMSPEED of NORMAL. Because status is saved at the end of each batch, a channel does not have any saved status until at least one batch has been transmitted.
- **Current** data consists of the common status fields and current-only status fields as noted in the syntax diagram. The data fields are continually updated as messages are sent/received.
- **Short** data consists of the STATUS current data item and the short status field as noted in the syntax diagram.

This method of operation has the following consequences:
- An inactive channel might not have any saved status - if it has never been current or has not yet reached a point where saved status is reset.
- The "common" data fields might have different values for saved and current status.
- A current channel always has current status and might have saved status.

Channels can either be current or inactive:

Current channels
: These are channels that have been started, or on which a client has connected, and that have not finished or disconnected normally. They might not yet have reached the point of transferring messages, or data, or even of establishing contact with the partner. Current channels have **current** status and might also have **saved** status.

  The term **Active** is used to describe the set of current channels that are not stopped.

Inactive channels
: These are channels that either:
  - Have not been started
  - On which a client has not connected
  - Have finished
  - Have disconnected normally

  (Note that if a channel is stopped, it is not yet considered to have finished normally - and is, therefore, still current.) Inactive channels have either **saved** status or no status at all.

There can be more than one instance of the same named receiver, requester, cluster-receiver, or server-connection channel current at the same time (the requester is acting as a receiver). This occurs if several senders, at different queue managers, each initiate a session with this receiver, using the same channel name. For channels of other types, there can only be one instance current at any time.

For all channel types, however, there can be more than one set of saved status information available for a channel name. At most one of these sets relates to a current instance of the channel, the rest relate to previously current instances. Multiple instances arise if different transmission queue names or connection names have been used with the same channel. This can happen in the following cases:
- At a sender or server:
  - If the same channel has been connected to by different requesters (servers only)
  - If the transmission queue name has been changed in the definition
  - If the connection name has been changed in the definition
- At a receiver or requester:

– If the same channel has been connected to by different senders or servers
– If the connection name has been changed in the definition (for requester channels initiating connection)

The number of sets that are displayed for a channel can be limited by using the XMITQ, CONNAME, and CURRENT parameters on the command.

(*generic-channel-name***)**
The name of the channel definition for which status information is to be displayed. A trailing asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions. A value is required for all channel types.

WHERE
Specify a filter condition to display status information for those channels that satisfy the selection criterion of the filter condition.

The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
The parameter to be used to display attributes for this DISPLAY command. However, you cannot use the following parameters as filter keywords: CHLDISP, CMDSCOPE, COMPRATE, COMPTIME, CURRENT, EXITTIME, JOBNAME (on z/OS), MCASTAT (on z/OS), MONITOR, NETTIME, SAVED, SHORT, XBATCHSZ, or XQTIME as filter keywords.

You cannot use CONNAME or XMITQ as filter keywords if you also use them to select channel status.

Status information for channels of a type for which the filter keyword is not valid is not displayed.

operator
This is used to determine whether a channel satisfies the filter value on the filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:
- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the CHLTYPE parameter), you can only use EQ or NE.

- A generic value. This is a character string with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. Use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed.

ALL Specify this to display all the status information for each relevant instance.

If SAVED is specified, this causes only common status information to be displayed, not current-only status information.

If this parameter is specified, any parameters requesting specific status information that are also specified have no effect; all the information is displayed.

CHLDISP

This parameter applies to z/OS only and specifies the disposition of the channels for which information is to be displayed, as used in the START and STOP CHANNEL commands, and **not** that set by QSGDISP for the channel definition. Values are:

ALL This is the default value and displays requested status information for private channels.

If there is a shared queue manager environment and the command is being executed on the queue manager where it was issued, or if CURRENT is specified, this option also displays the requested status information for shared channels.

PRIVATE

Display requested status information for private channels.

SHARED

Display requested status information for shared channels. This is allowed only if there is a shared queue manager environment, and either:

- CMDSCOPE is blank or the local queue manager
- CURRENT is specified

CHLDISP displays the following values:

PRIVATE

The status is for a private channel.

SHARED

The status is for a shared channel.

FIXSHARED

The status is for a shared channel, tied to a specific queue manager.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which it was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active

queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

Note: See Table 1, Table 2, and Table 3 for the permitted combinations of CHLDISP and CMDSCOPE.

CONNAME(*connection-name***)**
The connection name for which status information is to be displayed, for the specified channel or channels.

This parameter can be used to limit the number of sets of status information that is displayed. If it is not specified, the display is not limited in this way.

The value returned for CONNAME might not be the same as in the channel definition, and might differ between the current channel status and the saved channel status. (Using CONNAME for limiting the number of sets of status is therefore not recommended.)

For example, when using TCP, if CONNAME in the channel definition:

- Is blank or is in "host name" format, the channel status value has the resolved IP address.
- Includes the port number, the current channel status value includes the port number (except on z/OS), but the saved channel status value does not.

For SAVED or SHORT status, this value could also be the queue manager name, or queue-sharing group name, of the remote system.

CURRENT
This is the default, and indicates that current status information as held by the channel initiator for current channels only is to be displayed.

Both common and current-only status information can be requested for current channels.

Short status information is not displayed if this parameter is specified.

SAVED
Specify this to display saved status information for both current and inactive channels.

Only common status information can be displayed. Short and current-only status information is not displayed for current channels if this parameter is specified.

SHORT
This indicates that short status information and the STATUS item for current channels only is to be displayed.

Other common status and current-only status information is not displayed for current channels if this parameter is specified.

MONITOR
Specify this to return the set of online monitoring parameters. These are COMPRATE, COMPTIME, EXITTIME, MONCHL, NETTIME, XBATCHSZ, XQMSGSA, and XQTIME. If you specify this parameter, any of the monitoring parameters that you request specifically have no effect; all monitoring parameters are still displayed.

XMITQ(*q-name***)**
The name of the transmission queue for which status information is to be displayed, for the specified channel or channels.

This parameter can be used to limit the number of sets of status information that is displayed. If it is not specified, the display is not limited in this way.

The following information is always returned, for each set of status information:

- The channel name

- The transmission queue name (for sender and server channels)
- The connection name
- The remote queue-manager, or queue-sharing group, name (only for current status, and for all channel types except server-connection channels)
- The remote partner application name (for server-connection channels)
- The type of status information returned (CURRENT, SAVED, or on z/OS only, SHORT)
- STATUS (except SAVED on z/OS)
- On z/OS, CHLDISP
- STOPREQ (only for current status)
- SUBSTATE

If no parameters requesting specific status information are specified (and the ALL parameter is not specified), no further information is returned.

If status information is requested that is not relevant for the particular channel type, this is not an error.

Summary attributes

When SUMMARY or TOTAL are added to the MQSC command DISPLAY CHSTATUS, the number of conversations is displayed as the CONVS attribute. The following attributes display a summary for either each channel when SUMMARY is specified, or for all the channels when TOTAL is specified.

ALL Specify this to display all the status information for each relevant instance. This attribute is the default value if no attributes are requested.

If SAVED is specified, this causes only common status information to be displayed, not current-only status information.

If this parameter is specified, any parameters requesting specific status information that are also specified have no effect; all the information is displayed.

CURCNV
The number of current conversations.

Common status

The following information applies to all sets of channel status, whether or not the set is current. Some of this information does not apply to server-connection channels.

CHLTYPE
The channel type. This is one of the following:

SDR A sender channel

SVR A server channel

RCVR A receiver channel

RQSTR
A requester channel

CLUSSDR
A cluster-sender channel

CLUSRCVR
A cluster-receiver channel

SVRCONN
A server-connection channel

CURLUWID
The logical unit of work identifier associated with the current batch, for a sending or a receiving channel.

For a sending channel, when the channel is in doubt it is the LUWID of the in-doubt batch.

For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt.

It is updated with the LUWID of the next batch when this is known.

This parameter does not apply to server-connection channels.

CURMSGS
For a sending channel, this is the number of messages that have been sent in the current batch. It is incremented as each message is sent, and when the channel becomes in doubt it is the number of messages that are in doubt.

For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt.

For a receiving channel, it is the number of messages that have been received in the current batch. It is incremented as each message is received.

The value is reset to zero, for both sending and receiving channels, when the batch is committed.

This parameter does not apply to server-connection channels.

CURSEQNO
For a sending channel, this is the message sequence number of the last message sent. It is updated as each message is sent, and when the channel becomes in doubt it is the message sequence number of the last message in the in-doubt batch.

For a saved channel instance, this parameter has meaningful information only if the channel instance is in doubt. However, the parameter value is still returned when requested, even if the channel instance is not in doubt.

For a receiving channel, it is the message sequence number of the last message that was received. It is updated as each message is received.

This parameter does not apply to server-connection channels.

INDOUBT
Whether the channel is currently in doubt.

This is only YES while the sending Message Channel Agent is waiting for an acknowledgment that a batch of messages that it has sent has been successfully received. It is NO at all other times, including the period during which messages are being sent, but before an acknowledgment has been requested.

For a receiving channel, the value is always NO.

This parameter does not apply to server-connection channels.

LSTLUWID
The logical unit of work identifier associated with the last committed batch of messages transferred.

This parameter does not apply to server-connection channels.

LSTSEQNO
Message sequence number of the last message in the last committed batch. This number is not incremented by nonpersistent messages using channels with a NPMSPEED of FAST.

This parameter does not apply to server-connection channels.

STATUS
Current status of the channel. This is one of the following:

BINDING
Channel is performing channel negotiation and is not yet ready to transfer messages.

INITIALIZING
The channel initiator is attempting to start a channel. On z/OS, this is displayed as INITIALIZI.

PAUSED
The channel is waiting for the message-retry interval to complete before retrying an MQPUT operation.

REQUESTING
A local requester channel is requesting services from a remote MCA.

RETRYING
A previous attempt to establish a connection has failed. The MCA will reattempt connection after the specified time interval.

RUNNING
The channel is either transferring messages at this moment, or is waiting for messages to arrive on the transmission queue so that they can be transferred.

STARTING
A request has been made to start the channel but the channel has not yet begun processing. A channel is in this state if it is waiting to become active.

STOPPED
This state can be caused by one of the following:

- Channel manually stopped

  A user has entered a stop channel command against this channel.

- Retry limit reached

  The MCA has reached the limit of retry attempts at establishing a connection. No further attempt will be made to establish a connection automatically.

A channel in this state can be restarted only by issuing the START CHANNEL command, or starting the MCA program in an operating-system dependent manner.

STOPPING
Channel is stopping or a close request has been received.

SWITCHING
The channel is switching transmission queues.

On z/OS, STATUS is not displayed if saved data is requested.

On distributed platforms, the value of the STATUS field returned in the saved data is the status of the channel at the time the saved status was written. Normally, the saved status value is RUNNING. To see the current status of the channel, the user can use the DISPLAY CHSTATUS CURRENT command.

Note: For an inactive channel, CURMSGS, CURSEQNO, and CURLUWID have meaningful information only if the channel is INDOUBT. However they are still displayed and returned if requested.

Current-only status

The following information applies only to current channel instances. The information applies to all channel types, except where stated.

BATCHES
Number of completed batches during this session (since the channel was started).

BATCHSZ
The batch size being used for this session.

This parameter does not apply to server-connection channels, and no values are returned; if specified on the command, this is ignored.

BUFSRCVD
Number of transmission buffers received. This includes transmissions to receive control information only.

BUFSSENT
Number of transmission buffers sent. This includes transmissions to send control information only.

BYTSRCVD
Number of bytes received during this session (since the channel was started). This includes control information received by the message channel agent.

BYTSSENT
Number of bytes sent during this session (since the channel was started). This includes control information sent by the message channel agent.

CHSTADA
Date when this channel was started (in the form yyyy-mm-dd).

CHSTATI
Time when this channel was started (in the form hh.mm.ss).

COMPHDR
The technique used to compress the header data sent by the channel. Two values are displayed:

- The default header data compression value negotiated for this channel.
- The header data compression value used for the last message sent. The header data compression value can be altered in a sending channels message exit. If no message has been sent, the second value is blank.

COMPMSG
The technique used to compress the message data sent by the channel. Two values are displayed:

- The default message data compression value negotiated for this channel.
- The message data compression value used for the last message sent. The message data compression value can be altered in a sending channels message exit. If no message has been sent, the second value is blank.

COMPRATE
The compression rate achieved displayed to the nearest percentage. Two values are displayed:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

These values are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING. If monitoring data is not being collected, or if no messages have been sent by the channel, the values are shown as blank.

A value is only displayed for this parameter if MONCHL is set for this channel.

COMPTIME
The amount of time per message, displayed in microseconds, spent during compression or decompression. Two values are displayed:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

These values are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING. If monitoring data is not being collected, or if no messages have been sent by the channel, the values are shown as blank.

A value is only displayed for this parameter if MONCHL is set for this channel.

CURSHCNV
The CURSHCNV value is blank for all channel types other than server-connection channels. For each instance of a server-connection channel, the CURSHCNV output gives a count of the number of conversations currently running over that channel instance.

A value of zero indicates that the channel is running as it did in versions of IBM WebSphere MQ earlier than Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Sharing conversations
- Client Asynchronous consumption

EXITTIME
Amount of time, displayed in microseconds, spent processing user exits per message. Two values are displayed:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel.

HBINT
The heartbeat interval being used for this session.

JOBNAME
Name of job currently serving the channel.

- On IBM i, Windows, UNIX and Linux systems, this is the concatenation of the process identifier and the thread identifier of the MCA program, displayed in hexadecimal.

This information is not available on z/OS. The parameter is ignored if specified.

You cannot use JOBNAME as a filter keyword on z/OS.

KAINT
The keepalive interval being used for this session. This is valid only on z/OS.

LOCLADDR
Local communications address for the channel. The value returned depends on the TRPTYPE of the channel (currently only TCP/IP is supported).

LONGRTS
Number of long retry wait start attempts left. This applies only to sender or server channels.

LSTMSGDA
Date when the last message was sent or MQI call was handled, see LSTMSGTI.

LSTMSGTI
Time when the last message was sent or MQI call was handled.

For a sender or server, this is the time the last message (the last part of it if it was split) was sent. For a requester or receiver, it is the time the last message was put to its target queue. For a server-connection channel, it is the time when the last MQI call completed.

In the case of a server-connection channel instance on which conversations are being shared, this is the time when the last MQI call completed on any of the conversations running on the channel instance.

MAXMSGL
The maximum message length being used for this session (valid only on z/OS).

MAXSHCNV
The MAXSHCNV value is blank for all channel types other than server-connection channels. For each instance of a server-connection channel, the MAXSHCNV output gives the negotiated maximum of the number of conversations that can run over that channel instance.

A value of zero indicates that the channel is running as it did in versions of IBM WebSphere MQ earlier than Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Sharing conversations
- Client asynchronous consumption

MCASTAT
Whether the Message Channel Agent is currently running. This is either "running" or "not running".

Note that it is possible for a channel to be in stopped state, but for the program still to be running.

This information is not available on z/OS. The parameter is ignored if specified.

You cannot use MCASTAT as a filter keyword on z/OS.

MCAUSER
The user ID used by the MCA. This can be the user ID set in the channel definition, the default user ID for message channels, a user ID transferred from a client if this is a server-connection channel, or a user ID specified by a security exit.

This parameter applies only to server-connection, receiver, requester, and cluster-receiver channels.

On server connection channels that share conversations, the MCAUSER field contains a user ID if all the conversations have the same MCA user ID value. If the MCA user ID in use varies across these conversations, the MCAUSER field contains a value of *.

The maximum length is 12 characters on z/OS; on other platforms, it is 64 characters.

MONCHL
Current level of monitoring data collection for the channel.

This parameter is also displayed when you specify the MONITOR parameter.

MSGS
Number of messages sent or received (or, for server-connection channels, the number of MQI calls handled) during this session (since the channel was started).

In the case of a server-connection channel instance on which conversations are being shared, this is the total number of MQI calls handled on all of the conversations running on the channel instance.

NETTIME
Amount of time, displayed in microseconds, to send a request to the remote end of the channel and receive a response. This time only measures the network time for such an operation. Two values are displayed:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values may indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING.

This parameter applies only to sender, server, and cluster-sender channels.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel.

NPMSPEED
The nonpersistent message handling technique being used for this session.

RAPPLTAG
The remote partner application name. This is the name of the client application at the remote end of the channel. This parameter applies only to server-connection channels.

RPRODUCT
The remote partner product identifier. This is the product identifier of the IBM WebSphere MQ code running at the remote end of the channel. If the remote product identifier is blank, the remote partner is at version 6 or earlier. The possible values are shown in Table 85.

Table 85. Product Identifier values

Product Identifier	Description
MQMM	Queue Manager (non z/OS Platform)
MQMV	Queue Manager on z/OS
MQCC	WebSphere MQ C client
MQNC	IBM WebSphere MQ client for HP Integrity NonStop Server
MQNM	WebSphere MQ .NET fully managed client
MQJB	WebSphere MQ Classes for JAVA
MQJM	WebSphere MQ Classes for JMS (normal mode)
MQJN	WebSphere MQ Classes for JMS (migration mode)
MQJU	Common Java interface to the MQI
MQXC	XMS client C/C++ (normal mode)
MQXD	XMS client C/C++ (migration mode)
MQXN	XMS client .NET (normal mode)
MQXM	XMS client .NET (migration mode)
MQXU	WebSphere MQ .NET XMS client (unmanaged/XA)
MQNU	WebSphere MQ .NET unmanaged client

RQMNAME
The queue manager name, or queue-sharing group name, of the remote system. This parameter does not apply to server-connection channels.

RVERSION
The remote partner version. This is the version of the IBM WebSphere MQ code running at the remote end of the channel. If the remote version is blank, the remote partner is at version 6 or earlier.

The remote version is displayed as **VVRRMMFF**, where

VV Version

RR Release

MM Maintenance level

FF Fix level

SHORTRTS
Number of short retry wait start attempts left. This applies only to sender or server channels.

SSLCERTI
The full Distinguished Name of the issuer of the remote certificate. The issuer is the Certificate Authority that issued the certificate.

The maximum length is 256 characters. This limit might mean that exceptionally long Distinguished Names are truncated.

SSLCERTU
The local user ID associated with the remote certificate. This is valid on z/OS only.

SSLKEYDA
Date on which the previous successful SSL secret key reset was issued.

SSLKEYTI
Time at which the previous successful SSL secret key reset was issued.

SSLPEER
Distinguished Name of the peer queue manager or client at the other end of the channel.

The maximum length is 256 characters. This limit might mean that exceptionally long Distinguished Names are truncated.

SSLRKEYS
Number of successful SSL key resets. The count of SSL secret key resets is reset when the channel instance ends.

STOPREQ
Whether a user stop request is outstanding. This is either YES or NO.

SUBSTATE
Action being performed by the channel when this command is issued. The following substates are listed in precedence order, starting with the substate of the highest precedence:

ENDBATCH
Channel is performing end-of-batch processing.

SEND A request has been made to the underlying communication subsystem to send some data.

RECEIVE
A request has been made to the underlying communication subsystem to receive some data.

SERIALIZE
Channel is serializing its access to the queue manager. Valid on z/OS only.

RESYNCH
Channel is resynchronizing with the partner.

HEARTBEAT
Channel is heartbeating with the partner.

SCYEXIT
Channel is running the security exit.

RCVEXIT
Channel is running one of the receive exits.

SENDEXIT
Channel is running one of the send exits.

MSGEXIT
Channel is running one of the message exits.

MREXIT
Channel is running the message retry exit.

CHADEXIT
Channel is running through the channel auto-definition exit.

NETCONNECT
A request has been made to the underlying communication subsystem to connect a partner machine.

SSLHANDSHK
Channel is processing an SSL handshake.

NAMESERVER
A request has been made to the name server.

MQPUT
A request has been made to the queue manager to put a message on the destination queue.

MQGET
A request has been made to the queue manager to get a message from the transmission queue (if this is a message channel) or from an application queue (if this is an MQI channel).

MQICALL
A MQ API call, other than MQPUT and MQGET, is being executed.

COMPRESS
Channel is compressing or extracting data.

Not all substates are valid for all channel types or channel states. There are occasions when no substate is valid, at which times a blank value is returned.

For channels running on multiple threads, this parameter displays the substate of the highest precedence.

XBATCHSZ
Size of the batches transmitted over the channel. Two values are displayed:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values might indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING.

This parameter does not apply to server-connection channels.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel.

XQMSGSA
Number of messages queued on the transmission queue available to the channel for MQGETs.

This parameter has a maximum displayable value of 999. If the number of messages available exceeds 999, a value of 999 is displayed.

On z/OS, if the transmission queue is not indexed by *CorrelId*, this value is shown as blank.

This parameter applies to cluster-sender channels only.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel.

XQTIME
The time, in microseconds, that messages remained on the transmission queue before being retrieved. The time is measured from when the message is put onto the transmission queue until it is retrieved to be sent on the channel and, therefore, includes any interval caused by a delay in the putting application.

Two values are displayed:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values might indicate a problem with your system. They are reset every time the channel is started and are displayed only when the STATUS of the channel is RUNNING.

This parameter applies only to sender, server, and cluster-sender channels.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONCHL is set for this channel.

Short status

The following information applies only to current channel instances.

QMNAME
The name of the queue manager that owns the channel instance.

DISPLAY CHSTATUS (MQTT):

Use the MQSC command DISPLAY CHSTATUS (MQTT) to display the status of one or more IBM WebSphere MQ Telemetry channels.

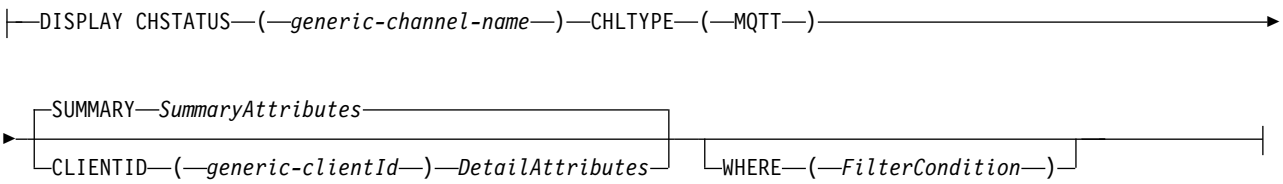
IBM i	UNIX and Linux	Windows	z/OS
	✓	✓	

Note: For the telemetry server, AIX is the only supported UNIX platform.

- Syntax diagram
- "Parameter descriptions for DISPLAY CHSTATUS" on page 626
- "Summary attributes" on page 628

Synonym: DIS CHS

DISPLAY CHSTATUS (MQTT):



SummaryAttributes:

DetailAttributes:

Note:

- The default behavior is for **RUNMQSC** to return a summary of the connections to the channel. If **CLIENTID** is specified then **RUNMQSC** returns details of each client connected to the channel.
- Either **CLIENTID**, **SUMMARY**, or neither may be specified, but not both at the same time.
- The **DISPLAY CHSTATUS** command for IBM WebSphere MQ Telemetry has the potential to return a far larger number of responses than if the command was run for a IBM WebSphere MQ channel. For this reason, the IBM WebSphere MQ Telemetry server does not return more responses than fit on the reply-to queue. The number of responses is limited to the value of MAXDEPTH parameter of the SYSTEM.MQSC.REPLY.QUEUE queue. When RUNMQSC processes a IBM WebSphere MQ Telemetry command that is truncated by the IBM WebSphere MQ Telemetry server, the AMQ8492 message is displayed specifying how many responses are returned based on the size of MAXDEPTH.

Parameter descriptions for DISPLAY CHSTATUS

You must specify the name of the channel for which you want to display status information. This parameter can be a specific channel name or a generic channel name. By using a generic channel name, you can display either the status information for all channels, or status information for one or more channels that match the specified name.

(*generic-channel-name***)**
The name of the channel definition for which status information is to be displayed. A trailing

asterisk (*) matches all channel definitions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all channel definitions. A value is required for all channel types.

WHERE

Specify a filter condition to display status information for those channels that satisfy the selection criterion of the filter condition.

The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword

The parameter to be used to display attributes for this DISPLAY command.

Status information for channels of a type for which the filter keyword is not valid is not displayed.

operator

This is used to determine whether a channel satisfies the filter value on the filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this operator to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this operator to display objects the attributes of which do not contain the specified item.

filter-value

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:

- An explicit value, that is a valid value for the attribute that is being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value SDR on the CHLTYPE parameter), you can use EQ or NE only.
- A generic value. This value is a character string with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. Use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed.

ALL Specify this parameter to display all the status information for each relevant instance.

If this parameter is specified, any parameters that request specific status information which are also specified have no effect; all the information is displayed.

Summary attributes

When SUMMARY or TOTAL are added to the MQSC command DISPLAY CHSTATUS, the number of conversations is displayed as the CONVS attribute. The following attributes display a summary for either each channel when SUMMARY is specified, or for all the channels when TOTAL is specified.

ALL Specify this parameter to display all the status information for each relevant instance. This attribute is the default value if no attributes are requested.

This parameter is valid for MQTT channels.

If this parameter is specified, any specified parameters that are requesting specific status information have no effect; and all the information is displayed.

CURCNV
The number of current conversations.

Client details mode

STATUS
The status of the client.

CONNAME
The name of the remote connection (IP address)

KAINT
The client's keep alive interval.

MCAUSER
The user ID being used by the channel.

MSGSNT
Number of messages sent by the client since it connected last.

MSGRCVD
Number of messages received by the client since it connected last.

INDOUBTIN
Number of in doubt, inbound messages to the client.

INDOUBTOUT
Number of in doubt, outbound messages to the client.

PENDING
Number of outbound pending messages.

LMSGDATE
Date last message was received or sent.

LMSGTIME
Time last message was received or sent.

CHLSDATE
Date channel started.

CHLSTIME
Time channel was started.

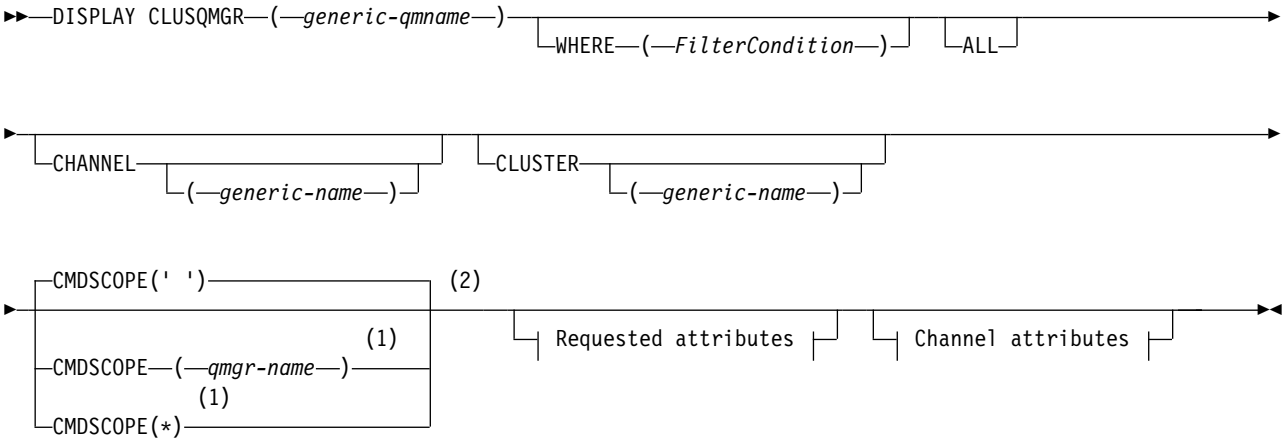
DISPLAY CLUSQMGR:

Use the MQSC command **DISPLAY CLUSQMGR** to display information about cluster channels for queue managers in a cluster.

UNIX and Linux	Windows
✓	✓

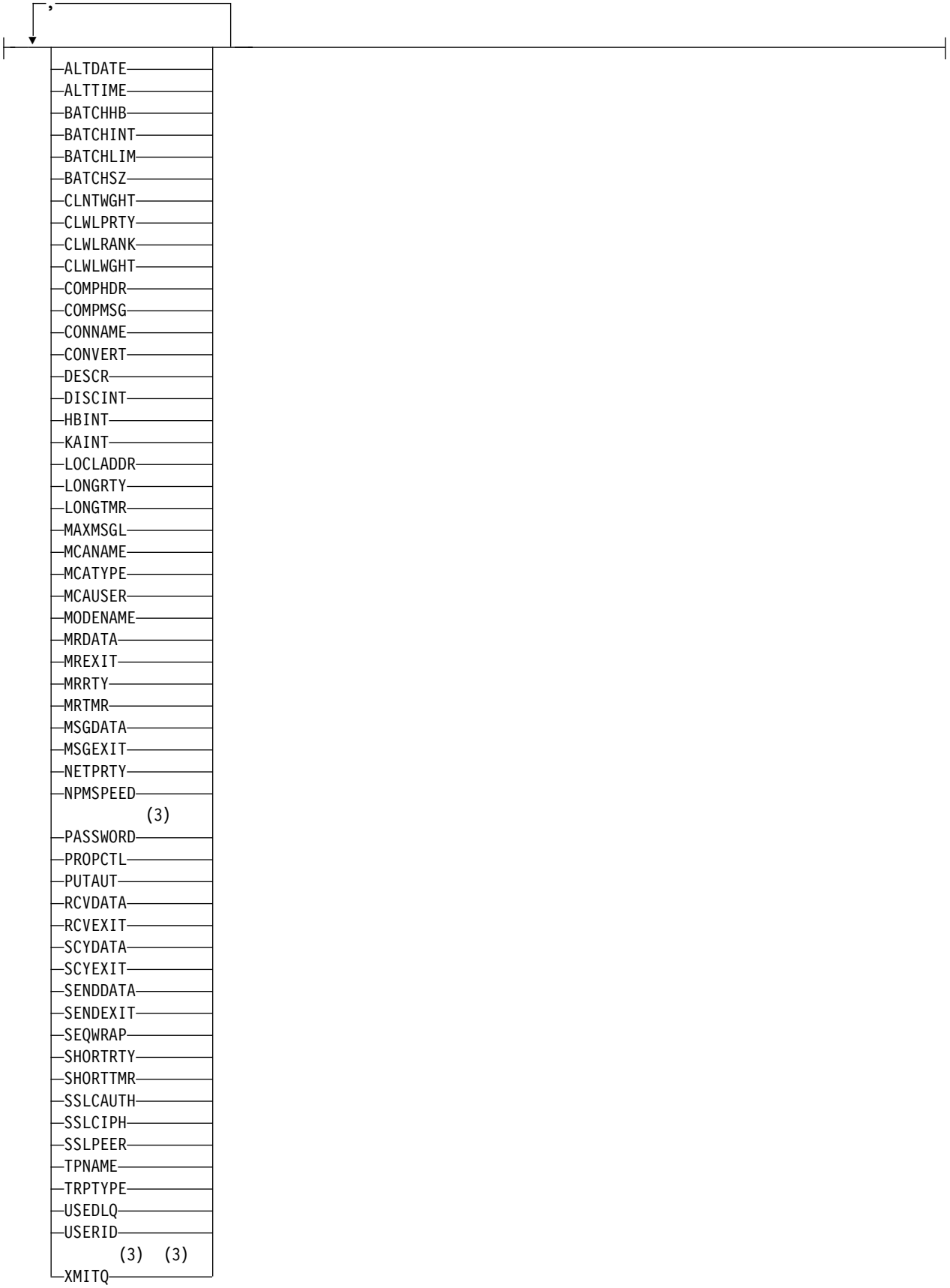
Synonym: DIS CLUSQMGR

DISPLAY CLUSQMGR



Requested attributes:

Channel attributes:


,
ALTDATE
ALTTIME
BATCHHB
BATCHINT
BATCHLIM
BATCHSZ
CLNTWGHT
CLWLPRTY
CLWLRANK
CLWLWGHT
COMPHDR
COMPMSG
CONNAME
CONVERT
DESCR
DISCINT
HBINT
KAINT
LOCLADDR
LONGRTY
LONGTMR
MAXMSGL
MCANAME
MCATYPE
MCAUSER
MODENAME
MRDATA
MREXIT
MRRTY
MRTMR
MSGDATA
MSGEXIT
NETPRTY
NPMSPEED
(3)
PASSWORD
PROPCTL
PUTAUT
RCVDATA
RCVEXIT
SCYDATA
SCYEXIT
SENDDATA
SENDEXIT
SEQWRAP
SHORTRTY
SHORTTMR
SSLCAUTH
SSLCIPH
SSLPEER
TPNAME
TRPTYPE
USEDLQ
USERID
(3) (3)
XMITQ

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Usage notes

Unlike the **DISPLAY CHANNEL** command, this command includes information about cluster channels that are auto-defined, and the status of cluster channels.

Note:

1. On UNIX systems, the command is valid only on AIX, HP-UX, Linux, and Solaris.
2. On z/OS, the command fails if the channel initiator is not started.

Parameter descriptions for DISPLAY CLUSQMGR

(*generic-qmgr-name***)**
The name of the cluster queue manager for which information is to be displayed.

A trailing asterisk "*" matches all cluster queue managers with the specified stem followed by zero or more characters. An asterisk "*" on its own specifies all cluster queue managers.

WHERE
Specify a filter condition to display only those cluster channels that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this **DISPLAY** command. However, you cannot use the **CMDSCOPE** or **MCANAME** parameters as filter keywords. You cannot use **CHANNEL** or **CLUSTER** as filter keywords if you use them to select cluster queue managers.

operator
The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use **CT** to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use **EX** to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use **CTG** to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a

filter-value. If the *filter-keyword* is a list, you can use **EXG** to display objects the attributes of which do not match the generic string.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, *filter-value* can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators **LT**, **GT, EQ, NE, LE,** , or **GE** only. If the attribute value is a value from a possible set of values, you can use only **EQ** or **NE**. For example, the value `STARTING` on the `STATUS` parameter.
- A generic value. *filter-value* is a character string. An example is `ABC*`. If the operator is **LK**, all items where the attribute value begins with the string, `ABC` in the example, are listed. If the operator is **NL**, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use **CT** or **EX** as the operator. For example, if the value `DEF` is specified with the operator **CT**, all items where one of the attribute values is `DEF` are listed. If it is generic, use **CTG** or **EXG** as the operator. If `ABC*` is specified with the operator **CTG**, all items where one of the attribute values begins with `ABC` are listed.

ALL Specify **ALL** to display all the parameters. If this parameter is specified, any parameters that are also requested specifically have no effect; all parameters are still displayed.

ALL is the default if you do not specify a generic name and do not request any specific parameters.

On z/OS **ALL** is also the default if you specify a filter condition using the **WHERE** parameter, but on other platforms, only requested attributes are displayed.

CHANNEL(*generic-name***)**
This is optional, and limits the information displayed to cluster channels with the specified channel name. The value can be a generic name.

CLUSTER(*generic-name***)**
This is optional, and limits the information displayed to cluster queue managers with the specified cluster name. The value can be a generic name.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

" The command is executed on the queue manager on which it was entered. " is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered. You can enter a different queue manager name, if you are using a queue-sharing group environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of * is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use **CMDSCOPE** as a filter keyword.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

Some parameters are relevant only for cluster channels of a particular type or types. Attributes that are not relevant for a particular type of channel cause no output, and do not cause an error.

CLUSDATE
The date on which the definition became available to the local queue manager, in the form `yyyy-mm-dd`.

CLUSTIME
The time at which the definition became available to the local queue manager, in the form `hh.mm.ss`.

DEFTYPE
How the cluster channel was defined:

CLUSSDR
As a cluster-sender channel from an explicit definition.

CLUSSDRA
As a cluster-sender channel by auto-definition alone.

CLUSSDRB
As a cluster-sender channel by auto-definition and an explicit definition.

CLUSRCVR
As a cluster-receiver channel from an explicit definition.

QMID
The internally generated unique name of the cluster queue manager.

QMTYPE
The function of the cluster queue manager in the cluster:

REPOS
Provides a full repository service.

NORMAL
Does not provide a full repository service.

STATUS
The status of the channel for this cluster queue manager is one of the following values:

STARTING
The channel was started and is waiting to become active.

BINDING
The channel is performing channel negotiation and is not yet ready to transfer messages.

INACTIVE
The channel is not active.

INITIALIZING
The channel initiator is attempting to start a channel. On z/OS, **INITIALIZING** is displayed as `INITIALIZI`.

RUNNING
The channel is either transferring messages at this moment, or is waiting for messages to arrive on the transmission queue so that they can be transferred.

STOPPING
The channel is stopping, or received a close request.

RETRYING
A previous attempt to establish a connection failed. The MCA attempts to connect again after the specified time interval.

PAUSED
The channel is waiting for the message-retry interval to complete before trying an MQPUT operation again.

STOPPED
This state can be caused by one of the following events:

- Channel manually stopped.

  A user entered a stop channel command for this channel.

- The number of attempts to establish a connection reached the maximum number of attempts allowed for the channel.

  No further attempt is made to establish a connection automatically.

A channel in this state can be restarted only by issuing the **START CHANNEL** command, or starting the MCA program in an operating-system dependent manner.

REQUESTING
A local requester channel is requesting services from a remote MCA.

SUSPEND
Specifies whether this cluster queue manager is suspended from the cluster or not (as a result of the **SUSPEND QMGR** command). The value of **SUSPEND** is either YES or NO.

XMITQ
The cluster transmission queue. The property is only available on platforms other than z/OS(r).

Channel parameters

ALTDATE
The date on which the definition or information was last altered, in the form yyyy-mm-dd

ALTTIME
The time at which the definition or information was last altered, in the form hh.mm.ss

BATCHHB
The batch heartbeat value being used.

BATCHINT
Minimum batch duration.

BATCHLIM
Batch data limit.

The limit of the amount of data that can be sent through a channel.

BATCHSZ
Batch size.

CLNTWGHT
The client channel weighting.

CLWLPRTY
The priority of the channel for the purposes of cluster workload distribution.

CLWLRANK
The rank of the channel for the purposes of cluster workload distribution.

CLWLWGHT
The weighting of the channel for the purposes of cluster workload distribution.

COMPHDR
The list of header data compression techniques supported by the channel.

COMPMSG
The list of message data compression techniques supported by the channel.

CONNAME
Connection name.

CONVERT
Specifies whether the sender converts application message data.

DESCR
Description.

DISCINT
Disconnection interval.

HBINT
Heartbeat interval.

KAINT
KeepAlive timing for the channel.

LOCLADDR
Local communications address for the channel.

LONGRTY
Limit of number of attempts to connect using the long duration timer.

LONGTMR
Long duration timer.

MAXMSGL
Maximum message length for channel.

MCANAME
Message channel agent name.

You cannot use MCANAME as a filter keyword.

MCATYPE
Specifies whether the message channel agent runs as a separate process or a separate thread.

MCAUSER
Message channel agent user identifier.

MODENAME
LU 6.2 mode name.

MRDATA
Channel message-retry exit user data.

MREXIT
Channel message-retry exit name.

MRRTY
Channel message-retry count.

MRTMR
Channel message-retry time.

MSGDATA
Channel message exit user data.

MSGEXIT
Channel message exit names.

NETPRTY
The priority for the network connection.

NPMSPEED
Nonpersistent message speed.

PASSWORD
Password for initiating LU 6.2 session (if nonblank, **PASSWORD** is displayed as asterisks).

PROPCTL
Message property control.

PUTAUT
Put authority.

RCVDATA
Channel receive exit user data.

RCVEXIT
Channel receive exit names.

SCYDATA
Channel security exit user data.

SCYEXIT
Channel security exit name.

SENDDATA
Channel send exit user data.

SENDEXIT
Channel send exit names.

SEQWRAP
Sequence number wrap value.

SHORTRTY
Limit of number of attempts to connect using the short duration timer.

SHORTTMR
Short duration timer.

SSLCAUTH
Specifies whether SSL client authentication is required.

SSLCIPH
Cipher specification for the SSL connection.

SSLPEER
Filter for the Distinguished Name from the certificate of the peer queue manager or client at the other end of the channel.

TRPTYPE
Transport type.

TPNAME
LU 6.2 transaction program name.

USEDLQ
Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

USERID
User identifier for initiating LU 6.2 session.

For more information about channel parameters, see “DEFINE CHANNEL” on page 437

DISPLAY COMMINFO:

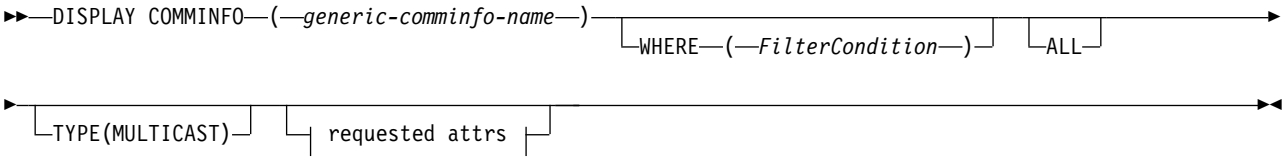
Use the MQSC command DISPLAY COMMINFO to display the attributes of a communication information object.

UNIX and Linux	Windows
✓	✓

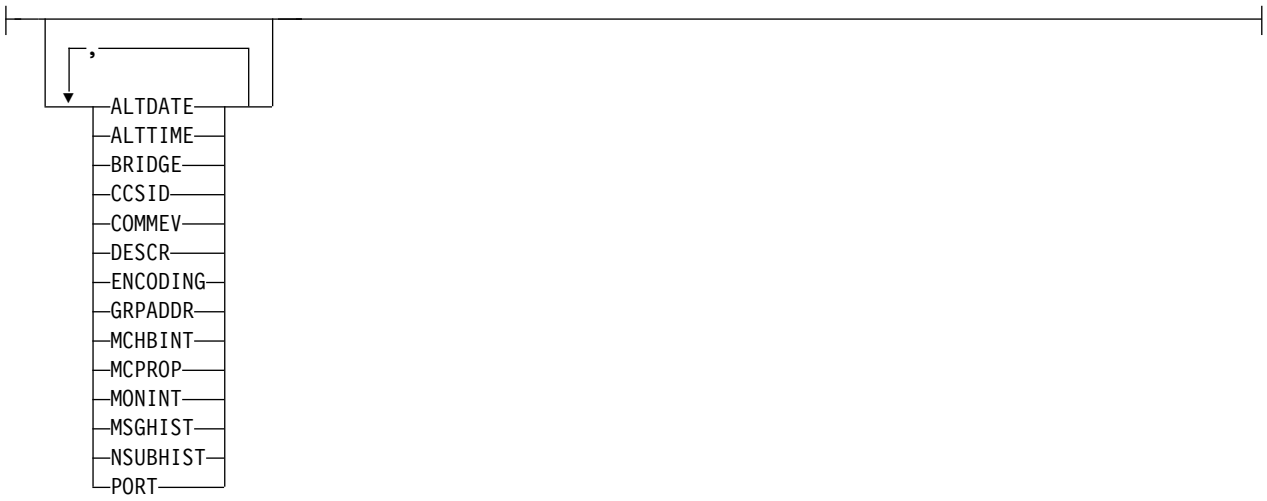
- Syntax diagram
- "Parameter descriptions for DISPLAY COMMINFO"
- "Requested parameters" on page 638

Synonym: DIS COMMINFO

DISPLAY COMMINFO



Requested attrs:



Parameter descriptions for DISPLAY COMMINFO

You must specify the name of the communication information object you want to display. This can be a specific communication information object name or a generic communication information object name. By using a generic communication information object name, you can display either:

- All communication information object definitions
- One or more communication information objects that match the specified name

(generic-comminfo-name)
The name of the communication information object definition to be displayed (see Rules for naming IBM WebSphere MQ objects). A trailing asterisk (*) matches all communication

information objects with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all communication information objects. The names must all be defined to the local queue manager.

WHERE
Specify a filter condition to display only those communication information object definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a communication information object definition satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value DISABLED on the COMMEV parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

TYPE Indicates the type of namelist to be displayed.

MULTICAST
Displays multicast communication information objects. This is the default.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the object names and TYPE parameters are displayed.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`

BRIDGE
Multicast bridging

CCSID
The coded character set identifier that messages are transmitted on.

COMMEV
Whether event messages are generated for Multicast.

DESCR(*string***)**
Description

ENCODING
The encoding that the messages are transmitted in.

GRPADDR
The group IP address or DNS name.

MCHBINT
Multicast heartbeat interval.

MCPROP
Multicast property control

MONINT
Monitoring frequency.

MSGHIST
The amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs (negative acknowledgments).

NSUBHIST
How much history a new subscriber joining a publication stream receives.

PORT The port number to transmit on.

DISPLAY CONN:

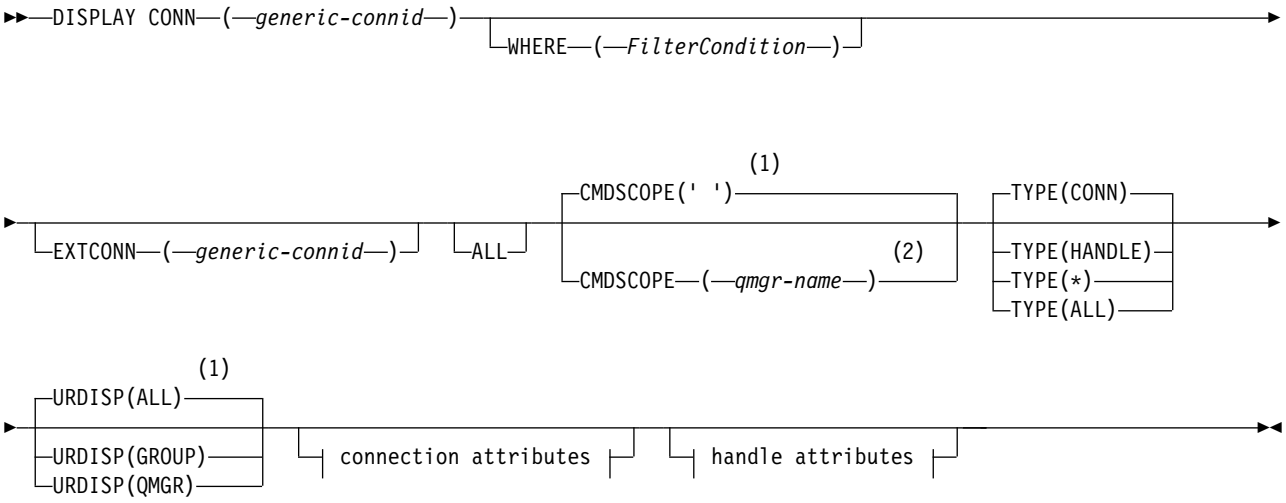
Use the MQSC command DISPLAY CONN to display connection information about the applications connected to the queue manager. This is a useful command because it enables you to identify applications with long-running units of work.

UNIX and Linux	Windows
✓	✓

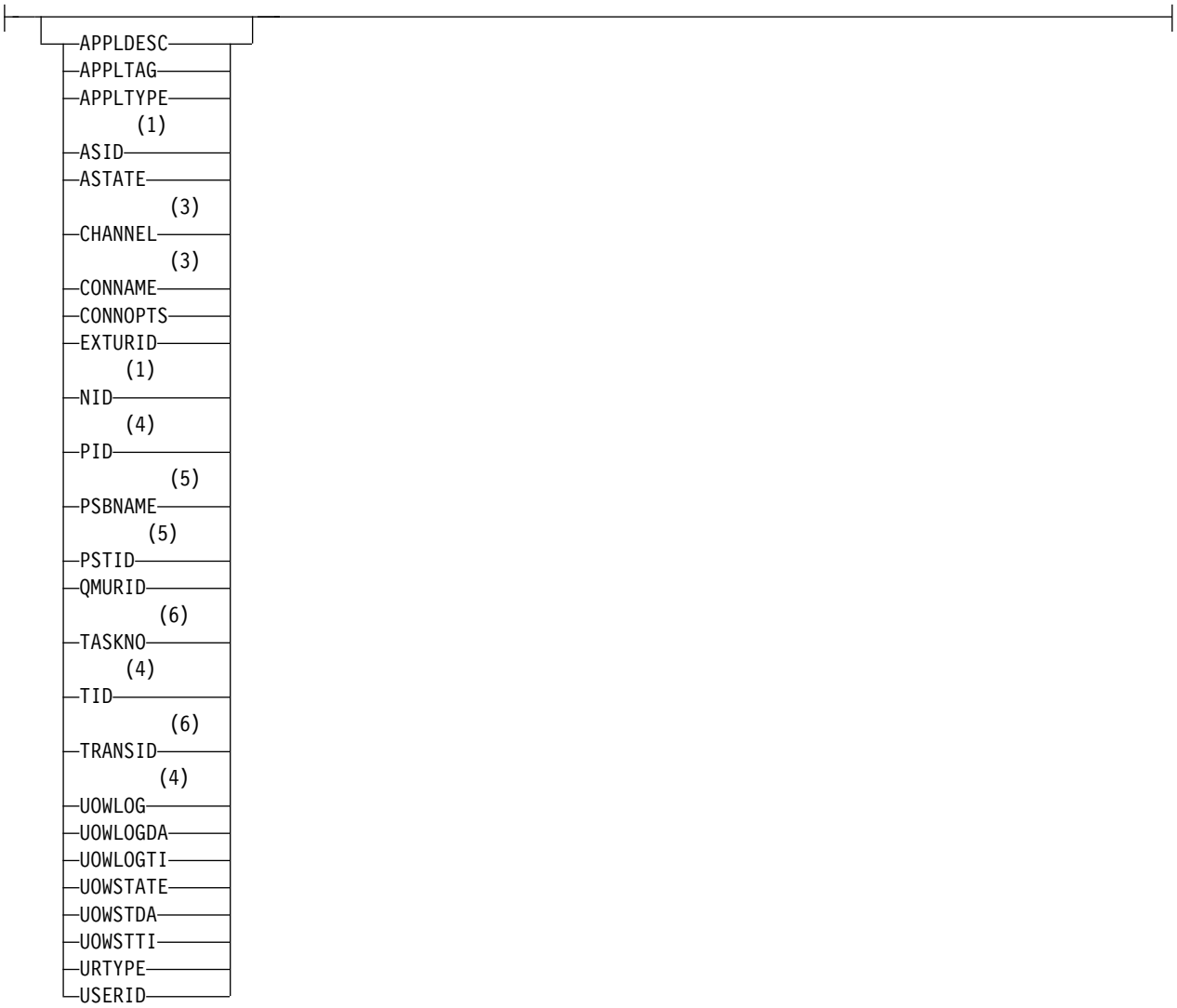
- Syntax diagram
- “Usage notes for DISPLAY CONN” on page 642
- “Parameter descriptions for DISPLAY CONN” on page 642
- “Connection attributes” on page 644
- “Handle attributes” on page 648
- “Full attributes” on page 652

Synonym: DIS CONN

DISPLAY CONN



Connection attributes:



Handle attributes:

Notes:

1 Valid only on z/OS.

2 Valid only when the queue manager is a member of a queue-sharing group.

3 Valid only when the connection is associated with a channel.

4 Not valid on z/OS.

5 IMS only.

6 CICS for z/OS only.

Usage notes for DISPLAY CONN

1. This command is issued internally by WebSphere MQ on z/OS when taking a checkpoint, and when the queue manager is starting and stopping, so that a list of units of work that are in doubt at the time is written to the z/OS console log.
2. The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed. On z/OS, these non-printable characters will be displayed as blanks. On distributed platforms using runmqsc, these non-printable characters will be displayed as dots.
3. The state of asynchronous consumers, ASTATE, reflects that of the server-connection proxy on behalf of the client application; it does not reflect the client application state.

Parameter descriptions for DISPLAY CONN

You must specify a connection for which you want to display information. This can be a specific connection identifier or a generic connection identifier. A single asterisk (*) can be used as a generic connection identifier to display information for all connections.

(*generic-connid***)**
The identifier of the connection definition for which information is to be displayed. A single asterisk (*) specifies that information for all connection identifiers is to be displayed.

When an application connects to WebSphere MQ, it is given a unique 24-byte connection identifier (ConnectionId). The value for CONN is formed by converting the last eight bytes of the ConnectionId to its 16 -character hexadecimal equivalent.

WHERE
Specify a filter condition to display only those connections that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE, EXTCONN, QSGDISP, TYPE, and EXTURID parameters as filter keywords.

operator
This is used to determine whether a connection satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item. You cannot use the CONNOPTS value MQCNO_STANDARD_BINDING with this operator.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item. You cannot use the CONNOPTS value MQCNO_STANDARD_BINDING with this operator.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value NONE on the UOWSTATE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string in the APPLTAG parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. Use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed.

ALL Specify this to display all the connection information of the requested type for each specified connection. This is the default if you do not specify a generic identifier, and do not request any specific parameters.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which it was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

EXTCONN
The value for EXTCONN is based on the first sixteen bytes of the ConnectionId converted to its 32 -character hexadecimal equivalent.

Connections are identified by a 24-byte connection identifier. The connection identifier comprises a prefix, which identifies the queue manager, and a suffix which identifies the connection to that

queue manager. By default, the prefix is for the queue manager currently being administered, but you can specify a prefix explicitly by using the EXTCONN parameter. Use the CONN parameter to specify the suffix.

When connection identifiers are obtained from other sources, specify the fully qualified connection identifier (both EXTCONN and CONN) to avoid possible problems related to non-unique CONN values.

Do not specify both a generic value for CONN and a non-generic value for EXTCONN.

You cannot use EXTCONN as a filter keyword.

TYPE Specifies the type of information to be displayed. Values are:

CONN
Connection information for the specified connection. On z/OS, this includes threads which may be logically or actually disassociated from a connection, together with those that are in-doubt and for which external intervention is needed to resolve them. These latter threads are those that DIS THREAD TYPE(INDOUBT) would show.

HANDLE
Information relating to any objects opened by the specified connection.

***** Display all available information relating to the connection.

ALL Display all available information relating to the connection.

URDISP
Specifies the unit of recovery disposition of connections to be displayed. Values are:

ALL Display all connections. This is the default option.

GROUP
Display only those connections with a GROUP unit of recovery disposition.

QMGR
Display only those connections with a QMGR unit of recovery disposition.

Connection attributes

If TYPE is set to CONN, the following information is always returned for each connection that satisfies the selection criteria, except where indicated:

- Connection identifier (CONN parameter)
- Type of information returned (TYPE parameter)

The following parameters can be specified for TYPE(CONN) to request additional information for each connection. If a parameter is specified that is not relevant for the connection, operating environment, or type of information requested, that parameter is ignored.

APPLDESC
A string containing a description of the application connected to the queue manager, where it is known. If the application is not recognized by the queue manager the description returned is blank.

APPLTAG
A string containing the tag of the application connected to the queue manager. It is one of the following:

- z/OS batch job name
- TSO USERID
- CICS APPLID
- IMS region name

- Channel initiator job name
- UNIX process

  Notes:

  – HP-UX On HP-UX if the process name exceeds 14 characters, only the first 14 characters are shown.

  – Linux Solaris On Linux and Solaris, if the process name exceeds 15 characters, only the first 15 characters are shown.

  – AIX On AIX, if the process name exceeds 28 characters, only the first 28 characters are shown.
- Windows process

  Note: This consists of the full program path and executable file name. If it is more than 28 characters long, only the last 28 characters are shown.
- Internal queue manager process name

APPLTYPE
A string indicating the type of the application that is connected to the queue manager. It is one of the following:

BATCH
Application using a batch connection

RRSBATCH
RRS-coordinated application using a batch connection

CICS CICS transaction

IMS IMS transaction

CHINIT
Channel initiator

OS400 An IBM i application

SYSTEM
Queue manager

SYSTEMEXT
Application performing an extension of function that is provided by the queue manager

UNIX A UNIX application

USER A user application

WINDOWSNT
A Windows application

ASID A 4-character address-space identifier of the application identified by APPLTAG. It distinguishes duplicate values of APPLTAG.

This parameter is returned only on z/OS when the APPLTYPE parameter does not have the value SYSTEM.

This parameter is valid only on z/OS.

ASTATE
The state of asynchronous consumption on this connection handle.

Possible values are:

SUSPENDED
An MQCTL call with the Operation parameter set to MQOP_SUSPEND has been issued against the connection handle so that so that asynchronous message consumption is temporarily suspended on this connection.

STARTED
An MQCTL call with the Operation parameter set to MQOP_START has been issued against the connection handle so that asynchronous message consumption can proceed on this connection.

STARTWAIT
An MQCTL call with the Operation parameter set to MQOP_START_WAIT has been issued against the connection handle so that asynchronous message consumption can proceed on this connection.

STOPPED
An MQCTL call with the Operation parameter set to MQOP_STOP has been issued against the connection handle so that asynchronous message consumption cannot currently proceed on this connection.

NONE
No MQCTL call has been issued against the connection handle. Asynchronous message consumption cannot currently proceed on this connection.

CHANNEL
The name of the channel that owns the connection. If there is no channel associated with the connection, this parameter is blank.

CONNAME
The connection name associated with the channel that owns the connection. If there is no channel associated with the connection, this parameter is blank.

CONNOPTS
The connect options currently in force for this application connection. Possible values are:

- MQCNO_HANDLE_SHARE_BLOCK
- MQCNO_HANDLE_SHARE_NO_BLOCK
- MQCNO_HANDLE_SHARE_NONE
- MQCNO_SHARED_BINDING
- MQCNO_STANDARD_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_QSG
- MQCNO_ACCOUNTING_Q_ENABLED
- MQCNO_ACCOUNTING_Q_DISABLED
- MQCNO_ACCOUNTING_MQI_ENABLED
- MQCNO_ACCOUNTING_MQI_DISABLED

You cannot use the value MQCNO_STANDARD_BINDING as a filter value with the CT and EX operators on the WHERE parameter.

EXTURID
The external unit of recovery identifier associated with this connection. Its format is determined by the value of URTYPE.

You cannot use EXTURID as a filter keyword.

NID Origin identifier, set only if the value of UOWSTATE is UNRESOLVED. This is a unique token identifying the unit of work within the queue manager. It is of the form `origin-node.origin-urid` where

- `origin-node` identifies the originator of the thread, except in the case where APPLTYPE is set to RRSBATCH, when it is omitted.
- `origin-urid` is the hexadecimal number assigned to the unit of recovery by the originating system for the specific thread to be resolved.

This parameter is valid only on z/OS.

PID Number specifying the process identifier of the application that is connected to the queue manager.

This parameter is not valid on z/OS.

PSBNAME
The 8-character name of the program specification block (PSB) associated with the running IMS transaction. You can use the PSBNAME and PSTID to purge the transaction using IMS commands. It is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value IMS.

PSTID
The 4-character IMS program specification table (PST) region identifier for the connected IMS region. It is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value IMS.

QMURID
The queue manager unit of recovery identifier. On z/OS, this is a 6 -byte log RBA, displayed as 12 hexadecimal characters. On platforms other than z/OS, this is an 8 -byte transaction identifier, displayed as `m.n` where `m` and `n` are the decimal representation of the first and last 4 bytes of the transaction identifier.

You can use QMURID as a filter keyword. On z/OS, you must specify the filter value as a hexadecimal string. On platforms other than z/OS, you must specify the filter value as a pair of decimal numbers separated by a period (.). You can only use the EQ, NE, GT, LT, GE, or LE filter operators. However, on z/OS, if log shunting has taken place, as indicated by message CSQR026I, instead of the RBA you have to use the URID from the message.

TASKNO
A 7-digit CICS task number. This number can be used in the CICS command "CEMT SET TASK(taskno) PURGE" to end the CICS task. This parameter is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value CICS.

TID Number specifying the thread identifier within the application process that has opened the specified queue.

This parameter is not valid on z/OS.

TRANSID
A 4-character CICS transaction identifier. This parameter is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value CICS.

UOWLOG
The file name of the extent to which the transaction associated with this connection first wrote.

This parameter is valid only on platforms other than z/OS.

UOWLOGDA
The date that the transaction associated with the current connection first wrote to the log.

UOWLOGTI
The time that the transaction associated with the current connection first wrote to the log.

UOWSTATE
The state of the unit of work. It is one of the following:

NONE
There is no unit of work.

ACTIVE
The unit of work is active.

PREPARED
The unit of work is in the process of being committed.

UNRESOLVED
The unit of work is in the second phase of a two-phase commit operation. WebSphere MQ holds resources on its behalf and external intervention is required to resolve it. This might be as simple as starting the recovery coordinator (such as CICS, IMS, or RRS) or it might involve a more complex operation such as using the RESOLVE INDOUBT command. The UNRESOLVED value can occur only on z/OS.

UOWSTDA
The date that the transaction associated with the current connection was started.

UOWSTTI
The time that the transaction associated with the current connection was started.

URTYPE
The type of unit of recovery as seen by the queue manager. It is one of the following:
- CICS (valid only on z/OS)
- XA
- RRS (valid only on z/OS)
- IMS (valid only on z/OS)
- QMGR

URTYPE identifies the EXTURID type and not the type of the transaction coordinator. When URTYPE is QMGR, the associated identifier is in QMURID (and not EXTURID).

USERID
The user identifier associated with the connection.

This parameter is not returned when APPLTYPE has the value SYSTEM.

Handle attributes

If TYPE is set to HANDLE, the following information is always returned for each connection that satisfies the selection criteria, except where indicated:
- Connection identifier (CONN parameter)
- Read ahead status (DEFREADA parameter)
- Type of information returned (TYPE parameter)
- Handle status (HSTATE)
- Object name (OBJNAME parameter)
- Object type (OBJTYPE parameter)

The following parameters can be specified for TYPE(HANDLE) to request additional information for each queue. If a parameter is specified that is not relevant for the connection, operating environment, or type of status information requested, that parameter is ignored.

ASTATE
The state of the asynchronous consumer on this object handle.

Possible values are:

ACTIVE
An MQCB call has set up a function to call back to process messages asynchronously and the connection handle has been started so that asynchronous message consumption can proceed.

INACTIVE
An MQCB call has set up a function to call back to process messages asynchronously but the connection handle has not yet been started, or has been stopped or suspended, so that asynchronous message consumption cannot currently proceed.

SUSPENDED
The asynchronous consumption callback has been suspended so that asynchronous message consumption cannot currently proceed on this object handle. This can be either because an MQCB call with Operation MQOP_SUSPEND has been issued against this object handle by the application, or because it has been suspended by the system. If it has been suspended by the system, as part of the process of suspending asynchronous message consumption the callback function will be called with the reason code that describes the problem resulting in suspension. This will be reported in the Reason field in the MQCBC structure that is passed to the callback function.

For asynchronous message consumption to proceed, the application must issue an MQCB call with the Operation parameter set to MQOP_RESUME.

SUSPTEMP
The asynchronous consumption callback has been temporarily suspended by the system so that asynchronous message consumption cannot currently proceed on this object handle. As part of the process of suspending asynchronous message consumption, the callback function will be called with the reason code that describes the problem resulting in suspension. This will be reported in the Reason field in the MQCBC structure passed to the callback function.

The callback function will be called again when asynchronous message consumption is resumed by the system, when the temporary condition has been resolved.

NONE
An MQCB call has not been issued against this handle, so no asynchronous message consumption is configured on this handle.

DEST The destination queue for messages that are published to this subscription. This parameter is only relevant for handles of subscriptions to topics. It is not returned for other handles.

DESTQMGR
The destination queue manager for messages that are published to this subscription. This parameter is relevant only for handles of subscriptions to topics. It is not returned for other handles. If DEST is a queue that is hosted on the local queue manager, this parameter will contain the local queue manager name. If DEST is a queue that is hosted on a remote queue manager, this parameter will contain the name of the remote queue manager.

HSTATE
The state of the handle.

Possible values are:

ACTIVE
An API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when an MQGET WAIT call is in progress.

If there is an MQGET SIGNAL outstanding, then this does not mean, by itself, that the handle is active.

INACTIVE
No API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when no MQGET WAIT call is in progress.

OBJNAME
The name of an object that the connection has open.

OBJTYPE
The type of the object that the connection has open. If this handle is that of a subscription to a topic, then the SUBID parameter identifies the subscription. You can then use the DISPLAY SUB command to find all the details about the subscription.

It is one of the following:
- QUEUE
- PROCESS
- QMGR
- STGCLASS (valid only on z/OS)
- NAMELIST
- CHANNEL
- AUTHINFO
- TOPIC

OPENOPTS
The open options currently in force for the connection for the object. This parameter is not returned for a subscription. Use the value in the SUBID parameter and the DISPLAY SUB command to find the details about the subscription.

Possible values are:

MQOO_INPUT_AS_Q_DEF
Open queue to get messages using queue-defined default.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_BROWSE
Open queue to browse messages.

MQOO_OUTPUT
Open queue or topic to put messages.

MQOO_INQUIRE
Open queue to inquire attributes.

MQOO_SET
Open queue to set attributes.

MQOO_BIND_ON_OPEN
Bind handle to destination when queue is found.

MQOO_BIND_NOT_FIXED
Do not bind to a specific destination.

MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved.

MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQOO_SET_IDENTITY_CONTEXT
Allow identity context to be set.

MQOO_SET_ALL_CONTEXT
Allow all context to be set.

MQOO_ALTERNATE_USER_AUTHORITY
Validate with specified user identifier.

MQOO_FAIL_IF_QUIESCING
Fail if queue manager is quiescing.

QSGDISP
Indicates the disposition of the object. It is valid on z/OS only. The value is one of the following:

QMGR
The object was defined with QSGDISP(QMGR).

COPY The object was defined with QSGDISP(COPY).

SHARED
The object was defined with QSGDISP(SHARED).

You cannot use QSGDISP as a filter keyword.

READA
The read ahead connection status.

Possible values are:

NO Read ahead of non-persistent messages is not enabled for this object.

YES Read ahead of non-persistent message is enabled for this object and is being used efficiently.

BACKLOG
Read ahead of non-persistent messages is enabled for this object. Read ahead is not being used efficiently because the client has been sent a large number of messages which are not being consumed.

INHIBITED
Read ahead was requested by the application but has been inhibited because of incompatible options specified on the first MQGET call.

SUBID
The internal, all-time unique identifier of the subscription. This parameter is relevant only for handles of subscriptions to topics. It is not returned for other handles.

Not all subscriptions show up in DISPLAY CONN; only those that have current handles open to the subscription show up. You can use the DISPLAY SUB command to see all subscriptions.

SUBNAME
The application's unique subscription name that is associated with the handle. This parameter is relevant only for handles of subscriptions to topics. It is not returned for other handles. Not all subscriptions will have a subscription name.

TOPICSTR
The resolved topic string. This parameter is relevant for handles with OBJTYPE(TOPIC). For any other object type, this parameter is not returned.

Full attributes

If TYPE is set to *, or ALL, both Connection attributes and Handle attributes are returned for each connection that satisfies the selection criteria.

DISPLAY ENTAUTH:

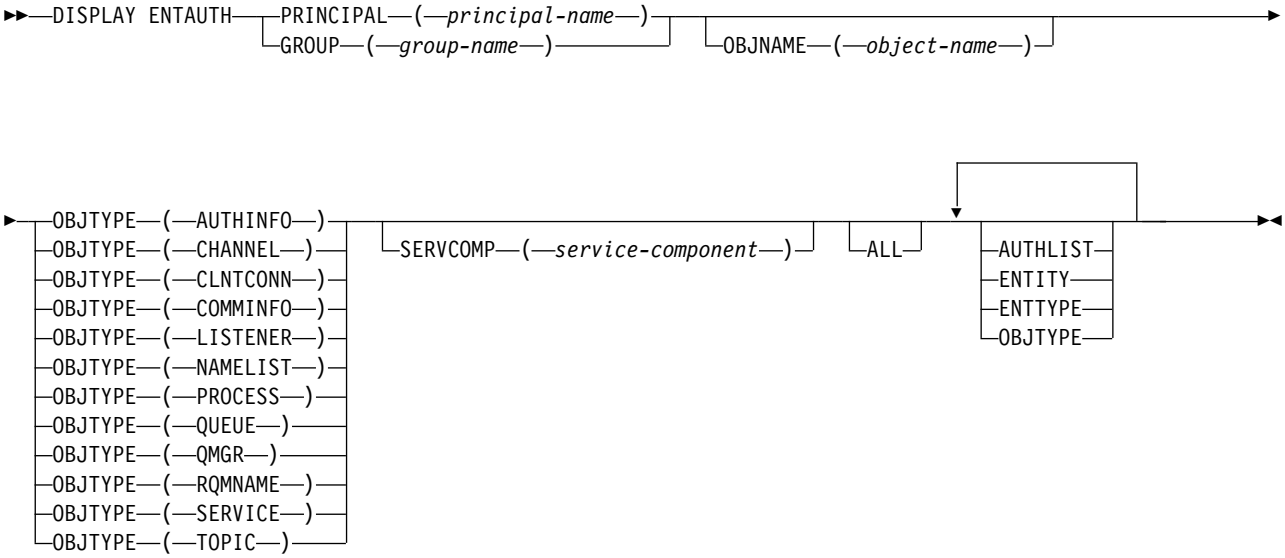
Use the MQSC command DISPLAY ENTAUTH to display the authorizations an entity has to a specified object.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions"
- "Requested parameters" on page 653

Synonym: DIS ENTAUTH

DISPLAY ENTAUTH



Parameter descriptions

PRINCIPAL(*principal-name*)
A principal name. This is the name of a user for whom to retrieve authorizations to the specified object. On IBM WebSphere MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

You must specify either PRINCIPAL or GROUP.

GROUP(*group-name*)
A group name. This is the name of the user group on which to make the inquiry. You can specify one name only and it must be the name of an existing user group.

For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

You must specify either PRINCIPAL or GROUP.

OBJNAME(*object-name*)
The name of the object or generic profile for which to display the authorizations.

This parameter is required unless the OBJTYPE parameter is QMGR. This parameter can be omitted if the OBJTYPE parameter is QMGR.

OBJTYPE
The type of object referred to by the profile. Specify one of the following values:

AUTHINFO
Authentication information record

CHANNEL
Channel

CLNTCONN
Client connection channel

COMMINFO
Communication information object

LISTENER
Listener

NAMELIST
Namelist

PROCESS
Process

QUEUE
Queue

QMGR
Queue manager

RQMNAME
Remote queue manager

SERVICE
Service

TOPIC
Topic

SERVCOMP(*service-component*)
The name of the authorization service for which information is to be displayed.

If you specify this parameter, it specifies the name of the authorization service to which the authorizations apply. If you omit this parameter, the inquiry is made to the registered authorization services in turn in accordance with the rules for chaining authorization services.

ALL
Specify this value to display all of the authorization information available for the entity and the specified profile.

Requested parameters

You can request the following information about the authorizations:

AUTHLIST
Specify this parameter to display the list of authorizations.

ENTITY
Specify this parameter to display the entity name.

ENTTYPE
Specify this parameter to display the entity type.

OBJTYPE
Specify this parameter to display the object type.

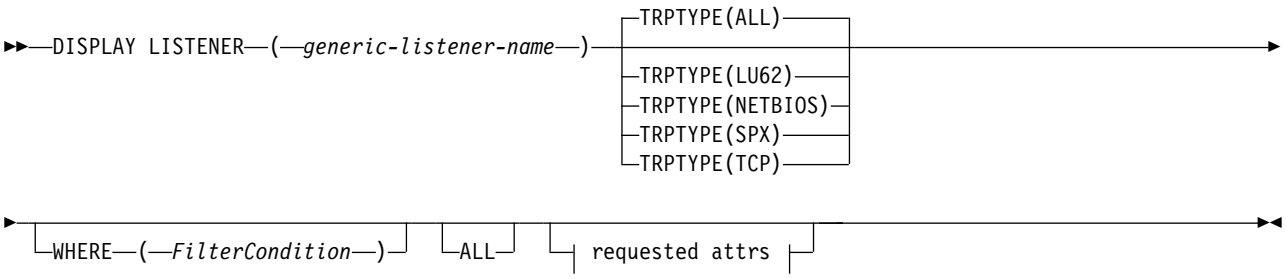
DISPLAY LISTENER:

Use the MQSC command DISPLAY LISTENER to display information about a listener.

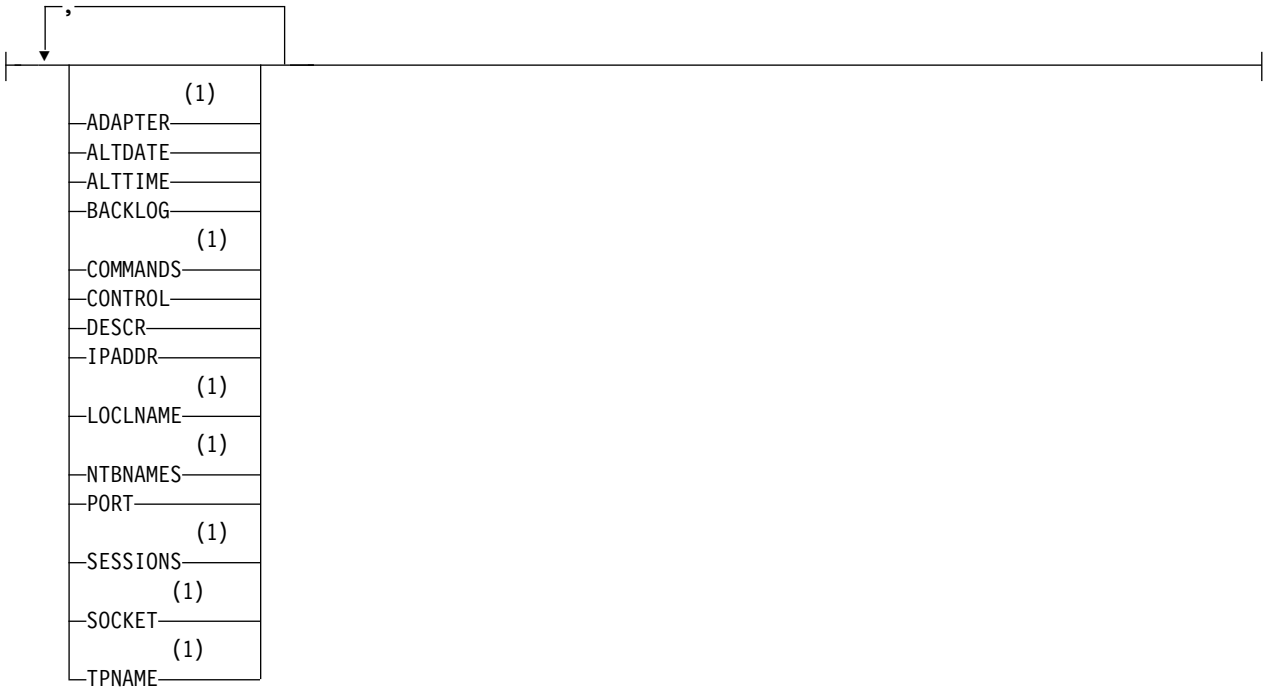
UNIX and Linux	Windows
✓	✓

Synonym: DIS LSTR

DISPLAY LISTENER



Requested attrs:



Notes:

1 Valid only on Windows.

Usage notes

The values displayed describe the current definition of the listener. If the listener has been altered since it was started, the currently running instance of the listener object may not have the same values as the current definition.

Keyword and parameter descriptions for DISPLAY LISTENER

You must specify a listener for which you want to display information. You can specify a listener by using either a specific listener name or a generic listener name. By using a generic listener name, you can display either:

- Information about all listener definitions, by using a single asterisk (*), or
- Information about one or more listeners that match the specified name.

(*generic-listener-name***)**
The name of the listener definition for which information is to be displayed. A single asterisk (*) specifies that information for all listener identifiers is to be displayed. A character string with an asterisk at the end matches all listeners with the string followed by zero or more characters.

TRPTYPE
Transmission protocol. If you specify this parameter, it must follow directly after the *generic-listener-name* parameter. If you do not specify this parameter, a default of ALL is assumed. Values are:

ALL This is the default value and displays information for all listeners.

LU62 Displays information for all listeners defined with a value of LU62 in their TRPTYPE parameter.

NETBIOS
Displays information for all listeners defined with a value of NETBIOS in their TRPTYPE parameter.

SPX Displays information for all listeners defined with a value of SPX in their TRPTYPE parameter.

TCP Displays information for all listeners defined with a value of TCP in their TRPTYPE parameter.

WHERE
Specify a filter condition to display information for those listeners that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a listener satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:
- An explicit value, that is a valid value for the attribute being tested.
- A generic value. This is a character string. with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Specify this to display all the listener information for each specified listener. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic identifier, and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

ADAPTER
The adapter number on which NetBIOS listens.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`.

BACKLOG
The number of concurrent connection requests that the listener supports.

COMMANDS
The number of commands that the listener can use.

CONTROL
How the listener is to be started and stopped:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR
Descriptive comment.

IPADDR
The listener's IP address.

LOCLNAME
The NetBIOS local name that the listener uses.

NTBNAMES
The number of names that the listener can use.

PORT The port number for TCP/IP.

SESSIONS
The number of sessions that the listener can use.

SOCKET
SPX socket.

TPNAME
The LU6.2 transaction program name.

For more information on these parameters, see “DEFINE LISTENER” on page 500.

DISPLAY LSSTATUS:

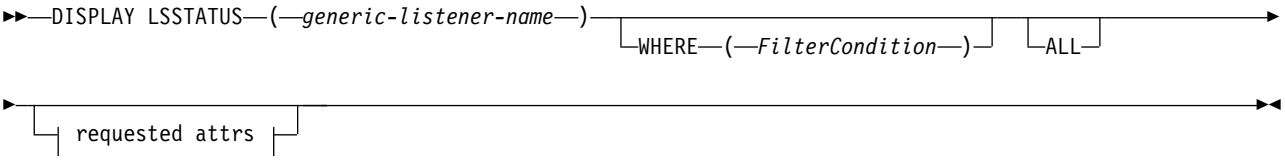
Use the MQSC command DISPLAY LSSTATUS to display status information for one or more listeners.

UNIX and Linux	Windows
✓	✓

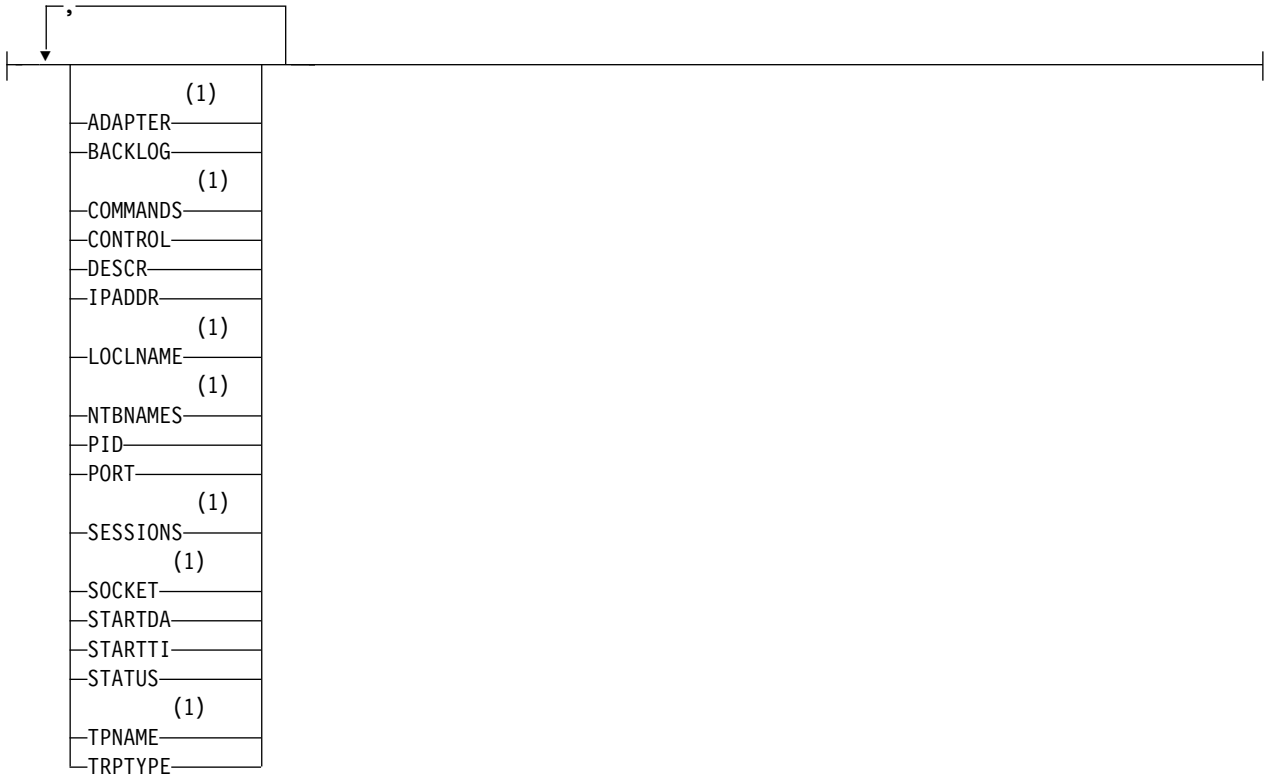
- Syntax diagram
- "Keyword and parameter descriptions for DISPLAY LSSTATUS" on page 659
- "Requested parameters" on page 659

Synonym: DIS LSSTATUS

DISPLAY LSSTATUS



Requested attrs:



Notes:

1 Valid only on Windows.

Keyword and parameter descriptions for DISPLAY LSSTATUS

You must specify a listener for which you want to display status information. You can specify a listener by using either a specific listener name or a generic listener name. By using a generic listener name, you can display either:

- Status information for all listener definitions, by using a single asterisk (*), or
- Status information for one or more listeners that match the specified name.

(*generic-listener-name***)**
The name of the listener definition for which status information is to be displayed. A single asterisk (*) specifies that information for all connection identifiers is to be displayed. A character string with an asterisk at the end matches all listeners with the string followed by zero or more characters.

WHERE
Specify a filter condition to display information for those listeners that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a listener satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:
- An explicit value, that is a valid value for the attribute being tested.
- A generic value. This is a character string. with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Display all the status information for each specified listener. This is the default if you do not specify a generic name, and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

ADAPTER
The adapter number on which NetBIOS listens.

BACKLOG
The number of concurrent connection requests that the listener supports.

CONTROL
How the listener is to be started and stopped:

MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by use of the START LISTENER and STOP LISTENER commands.

QMGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR
Descriptive comment.

IPADDR
The listener's IP address.

LOCLNAME
The NetBIOS local name that the listener uses.

NTBNAMES
The number of names that the listener can use.

PID The operating system process identifier associated with the listener.

PORT The port number for TCP/IP.

SESSIONS
The number of sessions that the listener can use.

SOCKET
SPX socket.

STARTDA
The date on which the listener was started.

STARTTI
The time at which the listener was started.

STATUS
The current status of the listener. It can be one of:

RUNNING
The listener is running.

STARTING
The listener is in the process of initializing.

STOPPING
The listener is stopping.

TPNAME
The LU6.2 transaction program name.

TRPTYPE
Transport type.

For more information on these parameters, see “DEFINE LISTENER” on page 500.

DISPLAY NAMELIST:

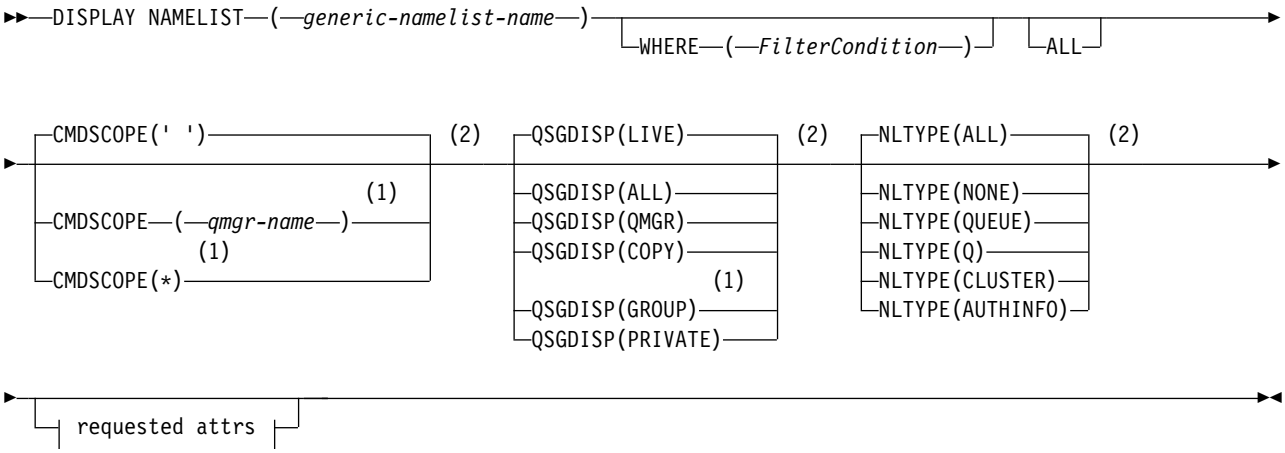
Use the MQSC command DISPLAY NAMELIST to display the names in a namelist.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- “Parameter descriptions for DISPLAY NAMELIST”
- “Requested parameters” on page 664

Synonym: DIS NL

DISPLAY NAMELIST



Requested attrs:

Notes:

1 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Parameter descriptions for DISPLAY NAMELIST

You must specify the name of the namelist definition you want to display. This can be a specific namelist name or a generic namelist name. By using a generic namelist name, you can display either:

- All namelist definitions
- One or more namelists that match the specified name

(*generic-namelist-name***)**
The name of the namelist definition to be displayed (see Rules for naming IBM WebSphere MQ objects). A trailing asterisk (*) matches all namelists with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all namelists.

WHERE
Specify a filter condition to display only those namelists that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE or QSGDISP parameters as filter keywords. You cannot use NLTYPE as a filter keyword if you also use it to select namelists.

operator
This is used to determine whether a namelist satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not contain the specified item.

CTG Contains an item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which match the generic string.

EXG Does not contain any item which matches a generic string that you provide as a *filter-value*. If the *filter-keyword* is a list, you can use this to display objects the attributes of which do not match the generic string.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:
- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value NONE on the NLTYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. The characters must be valid for the attribute you are testing. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

- An item in a list of values. The value can be explicit or, if it is a character value, it can be explicit or generic. If it is explicit, use CT or EX as the operator. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If it is generic, use CTG or EXG as the operator. If ABC* is specified with the operator CTG, all items where one of the attribute values begins with ABC are listed.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all the parameters are displayed.

This is the default if you do not specify a generic name, and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

CMDSCOPE

This parameter specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

QSGDISP

Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

In a shared queue manager environment, use

```
DISPLAY NAMELIST(name) CMDSCOPE(*) QSGDISP(ALL)
```

to list ALL objects matching

```
name
```

in the queue-sharing group without duplicating those in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP

Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). Note that QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

NLTYPE
Indicates the type of namelist to be displayed.

This parameter is valid only on z/OS.

ALL
Displays namelists of all types. This is the default.

NONE
Displays namelists of type NONE.

QUEUE or Q
Displays namelists that hold lists of queue names.

CLUSTER
Displays namelists that are associated with clustering.

AUTHINFO
Displays namelists that contain lists of authentication information object names.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the object names, and, on z/OS, their NLTYPEs and QSGDISP are displayed.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`

DESCR
Description

NAMCOUNT
Number of names in the list

NAMES
List of names

See "DEFINE NAMELIST" on page 503 for more information about the individual parameters.

DISPLAY PROCESS:

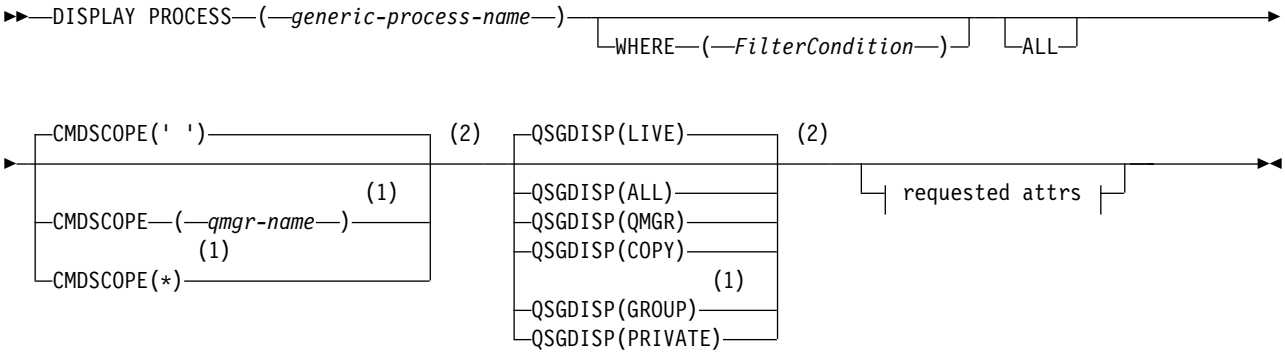
Use the MQSC command DISPLAY PROCESS to display the attributes of one or more WebSphere MQ processes.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for DISPLAY PROCESS"
- "Requested parameters" on page 667

Synonym: DIS PRO

DISPLAY PROCESS



Requested attrs:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

Parameter descriptions for DISPLAY PROCESS

You must specify the name of the process you want to display. This can be a specific process name or a generic process name. By using a generic process name, you can display either:

- All process definitions

- One or more processes that match the specified name

(generic-process-name)
The name of the process definition to be displayed (see Rules for naming IBM WebSphere MQ objects). A trailing asterisk (*) matches all processes with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all processes. The names must all be defined to the local queue manager.

WHERE
Specify a filter condition to display only those process definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE or QSGDISP parameters as filter keywords.

operator
This is used to determine whether a process definition satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:
- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value DEF on the APPLTYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

ALL Specify this to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

On AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS, this is the default if you do not specify a generic name and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

QSGDISP
Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(LIVE) is specified or defaulted, or if QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). Note that QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

The default, if no parameters are specified (and the ALL parameter is not specified) is that the object names and, on z/OS only, QSGDISP are displayed.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`

APPLICID
Application identifier

APPLTYPE
Application type. In addition to the values listed for this parameter in "Parameter descriptions for DEFINE PROCESS" on page 508, the value SYSTEM can be displayed. This indicates that the application type is a queue manager.

DESCR
Description

ENVRDATA
Environment data

USERDATA
User data

See "DEFINE PROCESS" on page 506 for more information about individual parameters.

DISPLAY PUBSUB:

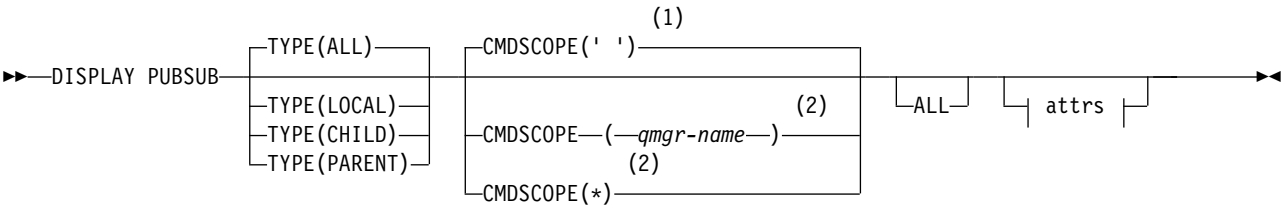
Use the MQSC command DISPLAY PUBSUB to display publish/subscribe status information for a queue manager.

UNIX and Linux	Windows
✓	✓

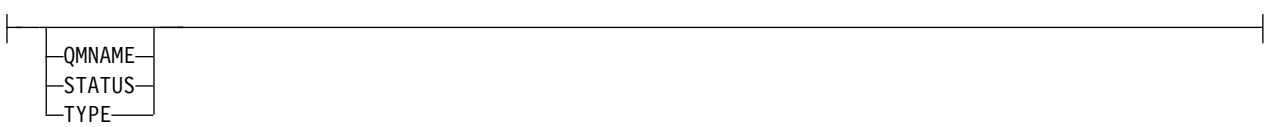
- Syntax diagram
- "Parameter descriptions for DISPLAY PUBSUB" on page 669
- "Returned parameters" on page 669

Synonym: None

DISPLAY PUBSUB



Attrs:



Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Parameter descriptions for DISPLAY PUBSUB

TYPE The type of publish/subscribe connections.

ALL Display the publish/subscribe status for this queue manager and for parent and child hierarchical connections.

CHILD
Display the publish/subscribe status for child connections.

LOCAL
Display the publish/subscribe status for this queue manager.

PARENT
Display the publish/subscribe status for the parent connection.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

Returned parameters

A group of parameters is returned, containing the attributes TYPE, QMNAME, and STATUS. This group is returned for the current queue manager if you set TYPE to LOCAL or ALL, for the parent queue manager if you set TYPE to PARENT or ALL, and for each child queue manager if you set TYPE to CHILD or ALL.

TYPE

CHILD
A child connection.

LOCAL
Information for this queue manager.

PARENT
The parent connection.

QMNAME
The name of the current queue manager or the remote queue manager connected as a parent or a child.

STATUS

The status of the publish/subscribe engine or the hierarchical connection. The publish/subscribe engine is initializing and is not yet operational. If the queue manager is a member of a cluster (has at least one CLUSRCVR defined), it remains in this state until the cluster cache is available. On WebSphere MQ for z/OS, this requires that the Channel Initiator is running.

When TYPE is LOCAL, the following values can be returned:

ACTIVE
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe using the application programming interface and the queues that are monitored by the queued publish/subscribe interface.

COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface. The queued publish/subscribe interface is not running. Therefore, any message that is put to the queues that are monitored by the queued publish/subscribe interface are not acted upon by IBM WebSphere MQ.

ERROR
The publish/subscribe engine has failed. Check your error logs to determine the reason for the failure.

INACTIVE
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted upon by IBM WebSphere MQ.

If inactive and you want to start the publish/subscribe engine use the command **ALTER QMGR PSMODE(ENABLED)**.

STARTING
The publish/subscribe engine is initializing and is not yet operational. If the queue manager is a member of a cluster, that is, it has at least one CLUSRCVR defined, it remains in this state until the cluster cache is available. On WebSphere MQ for z/OS, this requires that the Channel Initiator is running.

STOPPING
The publish/subscribe engine is stopping.

When TYPE is PARENT, the following values can be returned:

ACTIVE
The connection with the parent queue manager is active.

ERROR
This queue manager is unable to initialize a connection with the parent queue manager because of a configuration error. A message is produced in the queue manager logs to indicate the specific error. If you receive error message AMQ5821 or on z/OS systems CSQT821E, possible causes include:

- Transmit queue is full.
- Transmit queue put is disabled.

If you receive error message AMQ5814 or on z/OS systems CSQT814E, take the following actions:

- Check that the parent queue manager is correctly specified.
- Ensure that broker is able to resolve the queue manager name of the parent broker.

To resolve the queue manager name, at least one of the following resources must be configured:

- A transmission queue with the same name as the parent queue manager name.
- A queue manager alias definition with the same name as the parent queue manager name.
- A cluster with the parent queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the parent queue manager name.
- A default transmission queue.

After you have set up the configuration correctly, modify the parent queue manager name to blank. Then set with the parent queue manager name.

REFUSED
The connection has been refused by the parent queue manager. This might be caused by the following:

- The parent queue manager already has a child queue manager with the same name as this queue manager.
- The parent queue manager has used the command RESET QMGR TYPE(PUBSUB) CHILD to remove this queue manager as one of its children.

STARTING
The queue manager is attempting to request that another queue manager become its parent.

If the parent status remains in STARTING without progressing to ACTIVE, take the following actions:

- Check that the sender channel to parent queue manager is running
- Check that the receiver channel from parent queue manager is running

STOPPING
The queue manager is disconnecting from its parent.

If the parent status remains in STOPPING, take the following actions:

- Check that the sender channel to parent queue manager is running
- Check that the receiver channel from parent queue manager is running

When TYPE is CHILD, the following values can be returned:

ACTIVE
The connection with the child queue manager is active.

ERROR
This queue manager is unable to initialize a connection with the child queue manager because of a configuration error. A message is produced in the queue manager logs to indicate the specific error. If you receive error message AMQ5821 or on z/OS systems CSQT821E, possible causes include:

- Transmit queue is full.
- Transmit queue put is disabled.

If you receive error message AMQ5814 or on z/OS systems CSQT814E, take the following actions:

- Check that the child queue manager is correctly specified.
- Ensure that broker is able to resolve the queue manager name of the child broker.

To resolve the queue manager name, at least one of the following resources must be configured:

- A transmission queue with the same name as the child queue manager name.
- A queue manager alias definition with the same name as the child queue manager name.
- A cluster with the child queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the child queue manager name.
- A default transmission queue.

After you have set up the configuration correctly, modify the child queue manager name to blank. Then set with the child queue manager name.

STARTING
Another queue manager is attempting to request that this queue manager become its parent.

If the child status remains in STARTING without progressing to ACTIVE, take the following actions:

- Check that the sender channel to child queue manager is running
- Check that the receiver channel from child queue manager is running

STOPPING
The queue manager is disconnecting.

If the child status remains in STOPPING, take the following actions:

- Check that the sender channel to child queue manager is running
- Check that the receiver channel from child queue manager is running

DISPLAY QMGR:

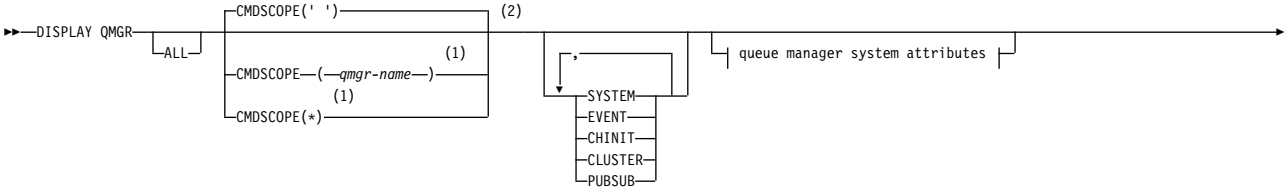
Use the MQSC command DISPLAY QMGR to display the queue manager parameters for this queue manager.

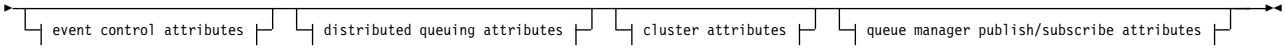
UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for DISPLAY QMGR" on page 674
- "Requested parameters" on page 675

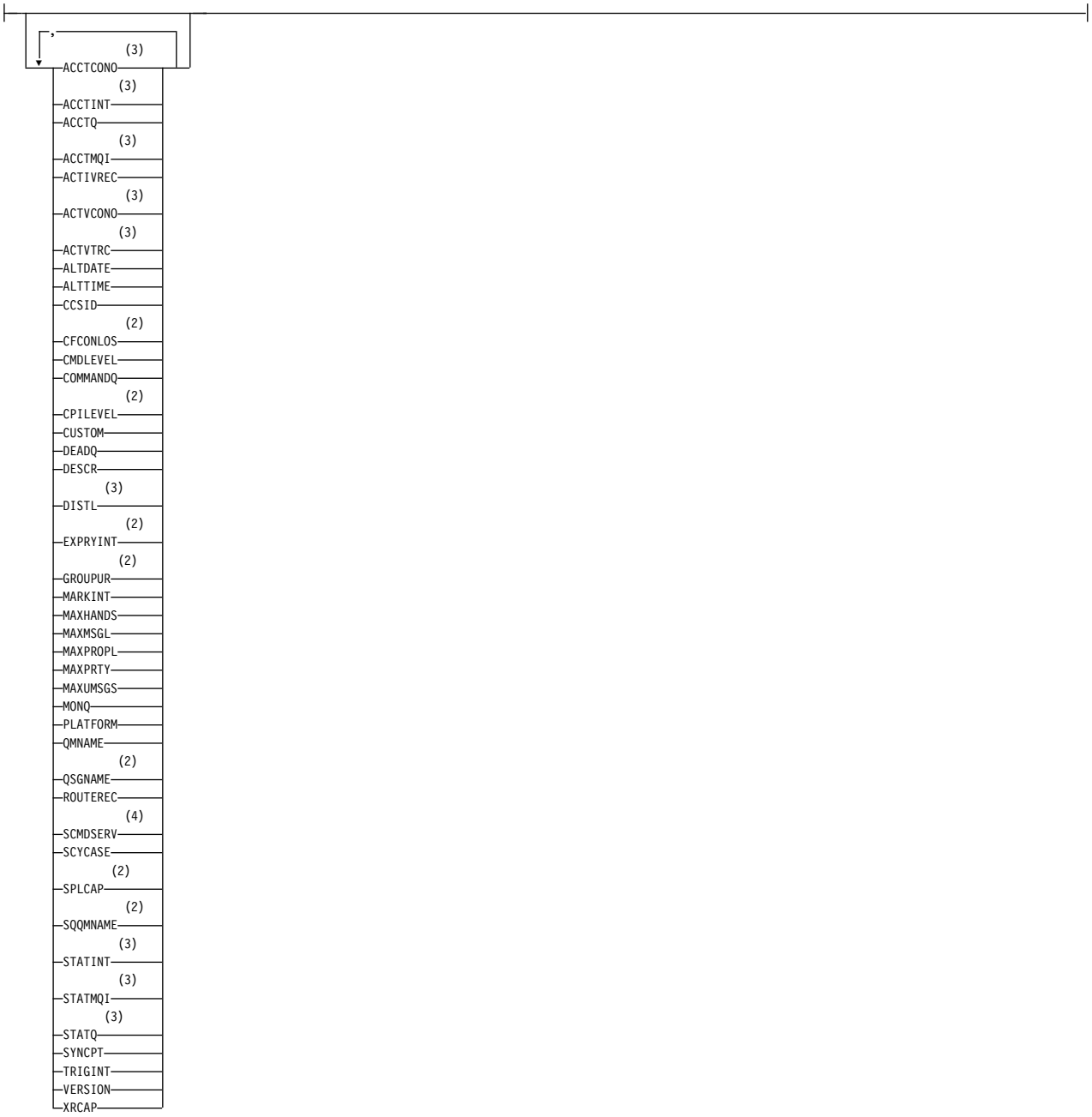
Synonym: DIS QMGR

DISPLAY QMGR





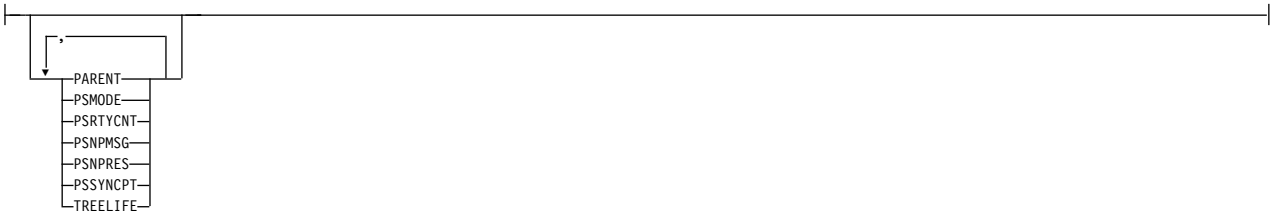
Queue manager system attributes:



Event control attributes:

Cluster attributes:

Queue manager publish/subscribe attributes:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Valid only on IBM i, UNIX, Linux, and Windows systems.

4 Not valid on z/OS.

Parameter descriptions for DISPLAY QMGR

ALL Specify this parameter to display all the parameters. If this parameter is specified, any parameters that are requested specifically are ineffective; all parameters are still displayed.

On AIX, HP-UX, Linux, IBM i, Solaris, and Windows, this parameter is the default if you do not request any specific parameters.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is run when the queue manager is a member of a queue-sharing group.

' ' The command is run on the queue manager on which it was entered. This command is the default value.

qmgr-name
The command is run on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is run on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of running this command is the same as entering the command on every queue manager in the queue-sharing group.

SYSTEM
Specify this parameter to display the set of queue manager system attributes that are available in the Queue manager system attrs list. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

EVENT
Specify this parameter to display the set of event control attributes that are available in the Event control attrs list. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

CHINIT
Specify this parameter to display the set of attributes relating to distributed queuing that are available in the Distributed queuing attrs list. You can also specify DQM to display the same set of attributes. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

CLUSTER
Specify this parameter to display the set of attributes relating to clustering that are available in the Cluster attrs list. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

PUBSUB
Specify this parameter to display the set of attributes relating to publish/subscribe that are available in the Queue manager pub/sub attrs list. See "Requested parameters" for information about these parameters.

If you specify this parameter, any request you make to display individual parameters within this set is ineffective.

Requested parameters

Note: If no parameters are specified (and the ALL parameter is not specified or defaulted), the queue manager name is returned.

You can request the following information for the queue manager:

ACCTCONO
Whether the settings of the ACCTQMQI and ACCTQ queue manager parameters can be overridden. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ACCTINT
The interval at which intermediate accounting records are written. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ACCTMQI
Whether accounting information is to be collected for MQI data. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ACCTQ
Whether accounting data collection is to be enabled for queues.

ACTCHL
The maximum number of channels that can be active at any time.

This parameter is valid only on z/OS.

ACTIVREC
Whether activity reports are to be generated if requested in the message.

ACTVCONO
Whether the settings of the ACTVTRC queue manager parameter can be overridden. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ACTVTRC
Whether WebSphere MQ MQI application activity tracing information is to be collected. See Setting ACTVTRC to control collection of activity trace information. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ADOPTCHK
Which elements are checked to determine whether an MCA is adopted when a new inbound channel is detected with the same name as an already active MCA.

This parameter is valid only on z/OS.

ADOPTMCA
Whether an orphaned MCA instance is to be restarted when a new inbound channel request matching the ADOPTCHK parameters is detected.

This parameter is valid only on z/OS.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`.

AUTHOREV
Whether authorization events are generated.

BRIDGEEV
On z/OS only, whether IMS Bridge events are generated.

CCSID
Coded character set identifier. This parameter applies to all character string fields defined by the application programming interface (API), including the names of objects, and the creation date and time of each queue. It does not apply to application data carried as the text of messages.

CERTVPOL
Specifies which SSL/TLS certificate validation policy is used to validate digital certificates received from remote partner systems. This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards. For more information about certificate validation policies, see Certificate validation policies in WebSphere MQ .

This parameter is valid on only UNIX, Linux, and Windows.

CFCONLOS
Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with CFCONLOS set to ASQMGR.

This parameter is valid only on z/OS.

CHAD
Whether auto-definition of receiver and server-connection channels is enabled. This parameter is not valid on z/OS.

CHADEV
Whether auto-definition events are enabled. This parameter is not valid on z/OS.

CHADEXIT
The name of the channel auto-definition exit.

CHIADAPS
The number of adapter subtasks to use to process IBM WebSphere MQ calls.

This parameter is valid only on z/OS.

CHIDISPS
The number of dispatchers to use for the channel initiator.

This parameter is valid only on z/OS.

CHISERVP
This field is reserved for IBM use only.

CHLAUTH
Whether channel authentication records are checked.

CHLEV
Whether channel events are generated.

CLWLEXIT
The name of the cluster workload exit.

CLWLDATA
The data passed to the cluster workload exit.

CLWLLEN
The maximum number of bytes of message data that is passed to the cluster workload exit.

This parameter is not valid on Linux.

CLWLMRUC
The maximum number of outbound cluster channels.

CLWLUSEQ
The behavior of MQPUTs for queues where CLWLUSEQ has a value of QMGR.

CMDEV
Whether command events are generated.

CMDLEVEL
Command level. This indicates the level of system control commands supported by the queue manager.

COMMANDQ
The name of the system-command input queue. Suitably authorized applications can put commands on this queue.

CONFIGEV
Whether configuration events are generated.

CPILEVEL
Reserved, this value has no significance.

CUSTOM
This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value in the form NAME(VALUE).

DEADQ
The name of the queue to which messages are sent if they cannot be routed to their correct destination (the dead-letter queue or undelivered-message queue). The default is blanks.

For example, messages are put on this queue when:

- A message arrives at a queue manager, destined for a queue that is not yet defined on that queue manager
- A message arrives at a queue manager, but the queue for which it is destined cannot receive it because, possibly:
  - The queue is full
  - The queue is inhibited for puts
  - The sending node does not have authority to put the message on the queue
- An exception message must be generated, but the queue named is not known to that queue manager

Note: Messages that have passed their expiry time are not transferred to this queue when they are discarded.

If the dead-letter queue is not defined, or full, or unusable for some other reason, a message that would have been transferred to it by a message channel agent is retained instead on the transmission queue.

If a dead-letter queue or undelivered-message queue is not specified, all blanks are returned for this parameter.

DEFCLXQ
The DEFCLXQ attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

SCTQ

All cluster-sender channels send messages from SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlID of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute DEFCLXQ was not present.

CHANNEL
Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

If the queue manager attribute, DEFCLXQ, is set to CHANNEL, the default configuration is changed to cluster-sender channels being associated with individual cluster transmission queues. The transmission queues are permanent-dynamic queues created from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Each transmission queue is associated with one cluster-sender channel. As one cluster-sender channel services a cluster transmission queue, the transmission queue contains messages for only one queue manager in one cluster. You can

configure clusters so that each queue manager in a cluster contains only one cluster queue. In this case, the message traffic from a queue manager to each cluster queue is transferred separately from messages to other queues.

DEFXMITQ
Default transmission queue name. This parameter is the transmission queue on which messages, destined for a remote queue manager, are put if there is no other suitable transmission queue defined.

DESCR
Description.

DISTL
Whether distribution lists are supported by the queue manager. This parameter is not valid on z/OS.

DNSGROUP
The name of the group that the TCP listener handling inbound transmissions for the queue-sharing group joins when using Workload Manager for Dynamic Domain Name Services support (WLM/DNS).

This parameter is valid only on z/OS.

DNSWLM
Whether the TCP listener that handles inbound transmissions for the queue-sharing group registers with WLM/DNS.

This parameter is valid only on z/OS.

EXPRYINT
On z/OS only, the approximate interval between scans for expired messages.

GROUPUR
On z/OS only, whether XA client applications are allowed to connect to this queue manager with a GROUP unit of recovery disposition.

IGQ On z/OS only, whether intra-group queuing is to be used.

IGQAUT
On z/OS only, displays the type of authority checking used by the intra-group queuing agent.

IGQUSER
On z/OS only, displays the user ID used by the intra-group queuing agent.

INHIBTEV
Whether inhibit events are generated.

IPADDRV
Whether to use an IPv4 or IPv6 IP address for a channel connection in ambiguous cases.

LOCALEV
Whether local error events are generated.

LOGGEREV
Whether recovery log events are generated. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

LSTRTMR
The time interval, in seconds, between attempts by IBM WebSphere MQ to restart the listener after an APPC or TCP/IP failure.

This parameter is valid only on z/OS.

LUGROUP
The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group.

This parameter is valid only on z/OS.

LUNAME
The name of the LU to use for outbound LU 6.2 transmissions.

This parameter is valid only on z/OS.

LU62ARM
The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. When automatic restart manager (ARM) restarts the channel initiator, the z/OS command SET APPC=xx is issued.

This parameter is valid only on z/OS.

LU62CHL
The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol. If the value of LU62CHL is zero, the LU 6.2 transmission protocol is not used.

This parameter is valid only on z/OS.

MARKINT
The mark browse interval in milliseconds.

MAXCHL
The maximum number of channels that can be current (including server-connection channels with connected clients).

This parameter is valid only on z/OS.

MAXHANDS
The maximum number of open handles that any one connection can have at any one time.

MAXMSGL
The maximum message length that can be handled by the queue manager. Individual queues or channels might have a smaller maximum than the value of this parameter.

MAXPROPL(*integer*)
The maximum length of property data in bytes that can be associated with a message.

This parameter is valid only on IBM i, z/OS, UNIX, Linux, and Windows systems.

MAXPRTY
The maximum priority. This value is 9.

MAXUMSGS
Maximum number of uncommitted messages within one sync point. The default value is 10000.

MAXUMSGS has no effect on IBM WebSphere MQ Telemetry. IBM WebSphere MQ Telemetry tries to batch requests to subscribe, unsubscribe, send, and receive messages from multiple clients into batches of work within a transaction.

MONACLS
Whether online monitoring data is to be collected for auto-defined cluster-sender channels, and, if so, the rate of data collection.

MONCHL
Whether online monitoring data is to be collected for channels, and, if so, the rate of data collection.

MONQ
Whether online monitoring data is to be collected for queues, and, if so, the rate of data collection.

OPORTMAX
The maximum value in the range of port numbers to be used when binding outgoing channels.

This parameter is valid only on z/OS.

OPORTMIN
The minimum value in the range of port numbers to be used when binding outgoing channels.

This parameter is valid only on z/OS.

PARENT
The name of the queue manager to which this queue manager is connected hierarchically as its child.

PERFMEV
Whether performance-related events are generated.

PLATFORM
The architecture of the platform on which the queue manager is running. The value of this parameter is MVS (for z/OS platforms), NSK, OS2, OS400, UNIX, or WINDOWSNT.

PSCLUS
Controls whether this queue manager participates in publish subscribe activity across any clusters in which it is a member. No clustered topic objects can exist in any cluster when modifying from ENABLED to DISABLED.

PSMODE
Controls whether the publish/subscribe engine and the queued publish/subscribe interface are running, and therefore controls whether applications can publish or subscribe by using the application programming interface and the queues that are monitored by the queued publish/subscribe interface.

PSNPMSG
If the queued publish/subscribe interface cannot process a non-persistent input message it might attempt to write the input message to the dead-letter queue (depending on the report options of the input message). If the attempt to write the input message to the dead-letter queue fails, and the MQRO_DISCARD_MSG report option was specified on the input message or PSNPMSG=DISCARD, the broker discards the input message. If PSNPMSG=KEEP is specified, the interface only discards the input message if the MQRO_DISCARD_MSG report option was set in the input message.

PSNPRES
If the queued publish/subscribe interface attempts to generate a response message in response to a non-persistent input message, and the response message cannot be delivered to the reply-to queue, this attribute indicates whether the interface tries to write the undeliverable message to the dead-letter queue or whether to discard the message.

PSRTYCNT
When the queued publish/subscribe interface fails to process a command message under sync point (for example a publish message that cannot be delivered to a subscriber because the subscriber queue is full and it is not possible to put the publication on the dead letter queue), the unit of work is backed out and the command tries this number of times again before the broker attempts to process the command message according to its report options instead.

PSSYNCPT
If this attribute is set to IFPER, when the queued publish/subscribe interface reads a publish or delete publication messages from a stream queue during normal operation then it specifies MQGMO_SYNCPOINT_IF_PERSISTENT. This value makes the queued pubsub daemon receive

non-persistent messages outside sync point. If the daemon receives a publication outside sync point, the daemon forwards that publication to subscribers known to it outside sync point.

QMID
The internally generated unique name of the queue manager.

QMNAME
The name of the local queue manager. See Rules for naming IBM WebSphere MQ objects.

QSGNAME
The name of the queue-sharing group to which the queue manager belongs, or blank if the queue manager is not a member of a queue-sharing group. You can use queue-sharing groups only onIBM WebSphere MQ for z/OS.

RCVTIME
The approximate length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state. The value of this parameter is the numeric value qualified by RCVTTYPE.

This parameter is valid only on z/OS.

RCVTMIN
The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state.

This parameter is valid only on z/OS.

RCVTTYPE
The qualifier to apply to the value in RCVTIME.

This parameter is valid only on z/OS.

REMOTEEV
Whether remote error events are generated.

REPOS
The name of a cluster for which this queue manager is to provide a repository manager service.

REPOSNL
The name of a list of clusters for which this queue manager is to provide a repository manager service.

ROUTEREC
Whether trace-route information is to be recorded if requested in the message.

SCHINIT
Whether the channel initiator is to be started automatically when the queue manager starts.

This parameter is not valid on z/OS.

SCMDSERV
Whether the command server is to be started automatically when the queue manager starts.

This parameter is not valid on z/OS.

SCYCASE
Whether the security profiles are uppercase or mixed case.

This parameter is valid only on z/OS.

If this parameter has been altered but the REFRESH SECURITY command has not yet been issued, the queue manager might not be using the case of profiles you expect. Use DISPLAY SECURITY to verify which case of profiles is actually in use.

SPLCAP

Indicates if WebSphere MQ Advanced Message Security (WebSphere MQ AMS) capabilities are available to the queue manager. If the WebSphere MQ AMS component is installed for the version of WebSphere MQ that the queue manager is running under, the attribute has a value `ENABLED (MQCAP_SUPPORTED)`. If the WebSphere MQ AMS component is not installed, the value is `DISABLED (MQCAP_NOT_SUPPORTED)`.

SQQMNAME
When a queue manager makes an MQOPEN call for a shared queue and the queue manager that is specified in the *ObjectQmgrName* parameter of the MQOPEN call is in the same queue-sharing group as the processing queue manager, the SQQMNAME attribute specifies whether the *ObjectQmgrName* is used or whether the processing queue manager opens the shared queue directly.

This parameter is valid only on z/OS.

SSLCRLNL
Indicates the namelist of AUTHINFO objects being used for the queue manager for certificate revocation checking.

SSLCRYP
Indicates the name of the parameter string being used to configure the cryptographic hardware present on the system. The PKCS #11 password appears as xxxxxx. This is valid only on UNIX, Linux, and Windows systems.

SSLEV
Whether SSL events are generated.

SSLFIPS
Whether only FIPS-certified algorithms are to be used if cryptography is executed in IBM WebSphere MQ rather than in the cryptographic hardware itself.

SSLKEYR
Indicates the name of the Secure Sockets Layer key repository.

SSLRKEYC
Indicates the number of bytes to be sent and received within an SSL conversation before the secret key is renegotiated.

SSLTASKS
On z/OS only, indicates the number of server subtasks to use for processing SSL calls.

STATACLS
Whether statistics data is to be collected for auto-defined cluster-sender channels, and, if so, the rate of data collection. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

STATCHL
Whether statistics data is to be collected for channels, and, if so, the rate of data collection. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

STATINT
The interval at which statistics monitoring data is written to the monitoring queue. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

STATMQI
Whether statistics monitoring data is to be collected for the queue manager. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

STATQ
Whether statistics data is to be collected for queues. This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

STRSTPEV
Whether start and stop events are generated.

SUITEB
Whether Suite B compliant cryptography is used. For more information about Suite B configuration and its effect on SSL and TLS channels, see NSA Suite B Cryptography in IBM WebSphere MQ .

SYNCPT
Whether sync point support is available with the queue manager.

TCPCHL
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol. If zero, the TCP/IP transmission protocol is not used.

This parameter is valid only on z/OS.

TCPKEEP
Whether the KEEPALIVE facility is to be used to check that the other end of the connection is still available. If it is unavailable, the channel is closed.

This parameter is valid only on z/OS.

TCPNAME
The name of the TCP/IP system that you are using.

This parameter is valid only on z/OS.

TCPSTACK
Whether the channel initiator uses only the TCP/IP address space specified in TCPNAME, or optionally binds to any selected TCP/IP address.

This parameter is valid only on z/OS.

TRAXSTR
Whether channel initiator trace starts automatically.

This parameter is valid only on z/OS.

TRAXTBL
The size, in megabytes, of the trace data space of the channel initiator.

This parameter is valid only on z/OS.

TREELIFE
The lifetime of non-administrative topics.

TRIGINT
The trigger interval.

VERSION
The version of the IBM WebSphere MQ installation, the queue manager is associated with. The version has the format `VVRRMMFF`:

`VV`: Version

`RR`: Release

`MM`: Maintenance level

`FF`: Fix level

XRCAP
Whether IBM WebSphere MQ Telemetry capability is supported by the queue manager.

For more details of these parameters, see “ALTER QMGR” on page 353.

DISPLAY QMSTATUS:

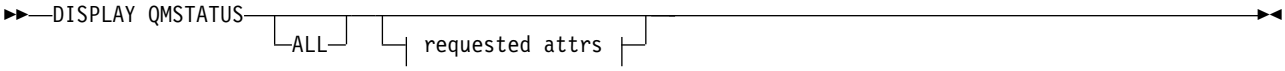
Use the MQSC command DISPLAY QMSTATUS to display status information associated with this queue manager.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for DISPLAY QMSTATUS"
- "Requested parameters"

Synonym: DIS QMSTATUS

DISPLAY QMSTATUS



Requested attrs:

Parameter descriptions for DISPLAY QMSTATUS

ALL Specify this parameter to display all the parameters. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This parameter is the default if you do not request any specific parameters.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

CHINIT
The status of the channel initiator reading SYSTEM.CHANNEL.INITQ. It is one of the following:

STOPPED
The channel initiator is not running.

STARTING
The channel initiator is in the process of initializing and is not yet operational.

RUNNING
The channel initiator is fully initialized and is running.

STOPPING
The channel initiator is stopping.

CMDSERV
The status of the command server. It is one of the following:

STOPPED
The command server is not running.

STARTING
The command server is in the process of initializing and is not yet operational.

RUNNING
The command server is fully initialized and is running.

STOPPING
The command server is stopping.

CONNS
The current number of connections to the queue manager.

CURRLOG
The name of the log extent being written to at the time that the DISPLAY QMSTATUS command is processed. If the queue manager is using circular logging, and this parameter is explicitly requested, a blank string is displayed.

INSTDESC
Description of the installation associated with the queue manager. This parameter is not valid on IBM i.

INSTNAME
Name of the installation associated with the queue manager. This parameter is not valid on IBM i.

INSTPATH
Path of the installation associated with the queue manager. This parameter is not valid on IBM i.

MEDIALOG
The name of the oldest log extent required by the queue manager to perform media recovery. If the queue manager is using circular logging, and this parameter is explicitly requested, a blank string is displayed.

QMNAME
The name of the queue manager. This parameter is always returned.

RECLOG
The name of the oldest log extent required by the queue manager to perform restart recovery. If the queue manager is using circular logging, and this parameter is explicitly requested, a blank string is displayed.

STATUS
The status of the queue manager. It is one of the following:

STARTING
The queue manager is in the process of initializing.

RUNNING
The queue manager is fully initialized and is running.

QUIESCING
The queue manager is quiescing.

STARTDA
The date on which the queue manager was started (in the form yyyy-mm-dd).

STARTTI
The time at which the queue manager was started (in the form hh.mm.ss).

DISPLAY QSTATUS:

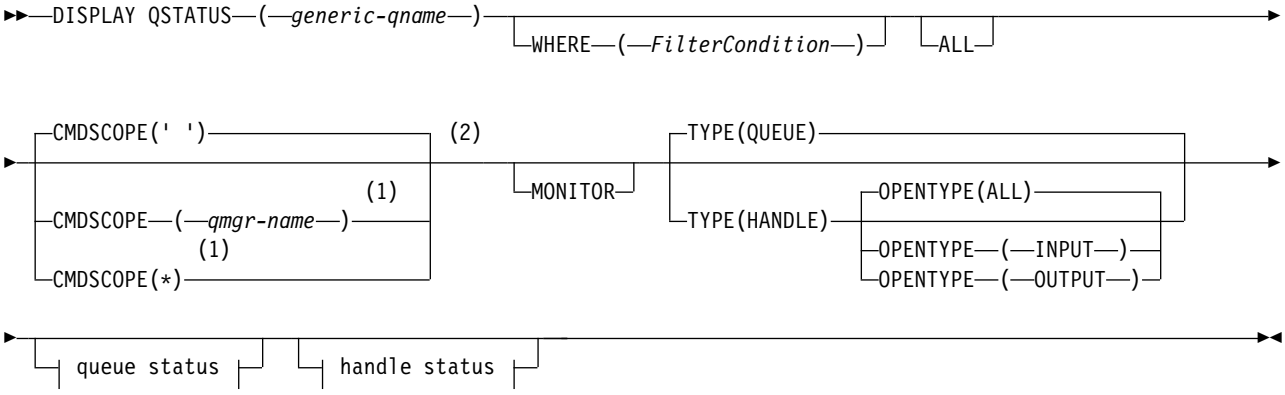
Use the MQSC command DISPLAY QSTATUS to display the status of one or more queues.

UNIX and Linux	Windows
✓	✓

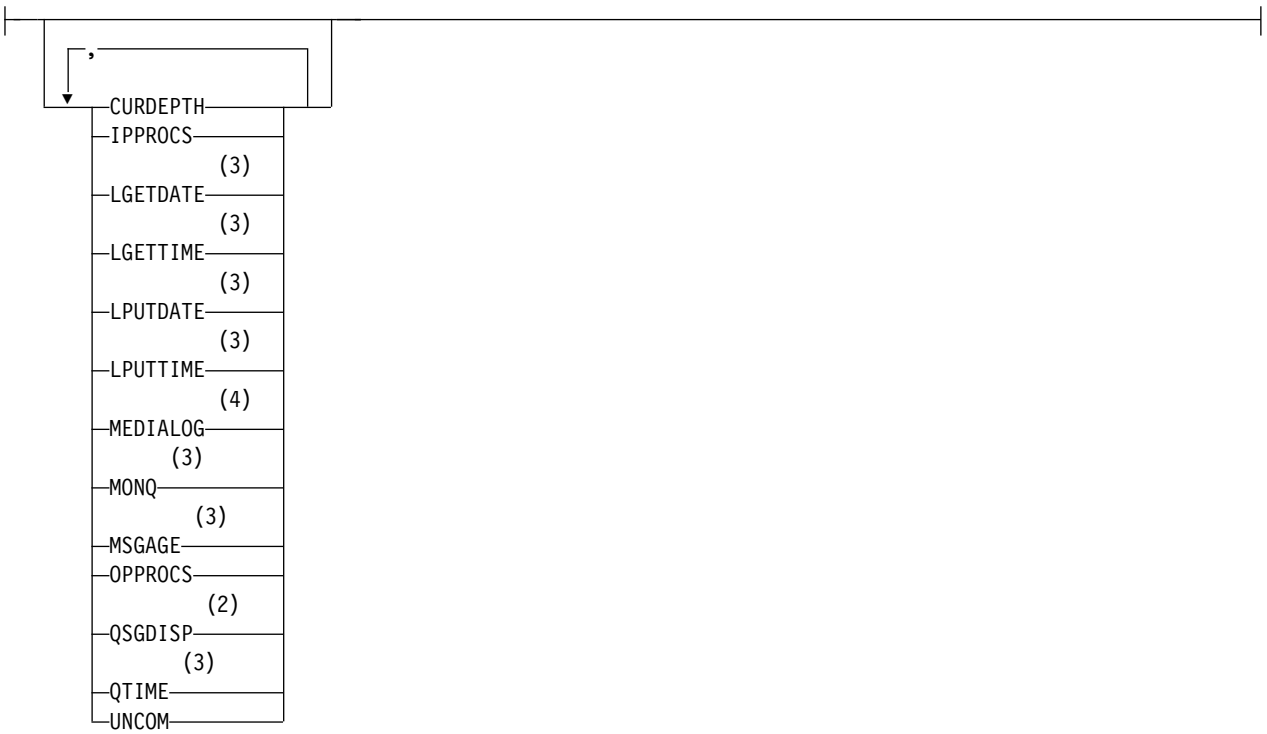
- Syntax diagram
- "Usage notes for DISPLAY QSTATUS" on page 689
- "Parameter descriptions for DISPLAY QSTATUS" on page 690
- "Queue status" on page 692
- "Handle status" on page 694

Synonym: DIS QS

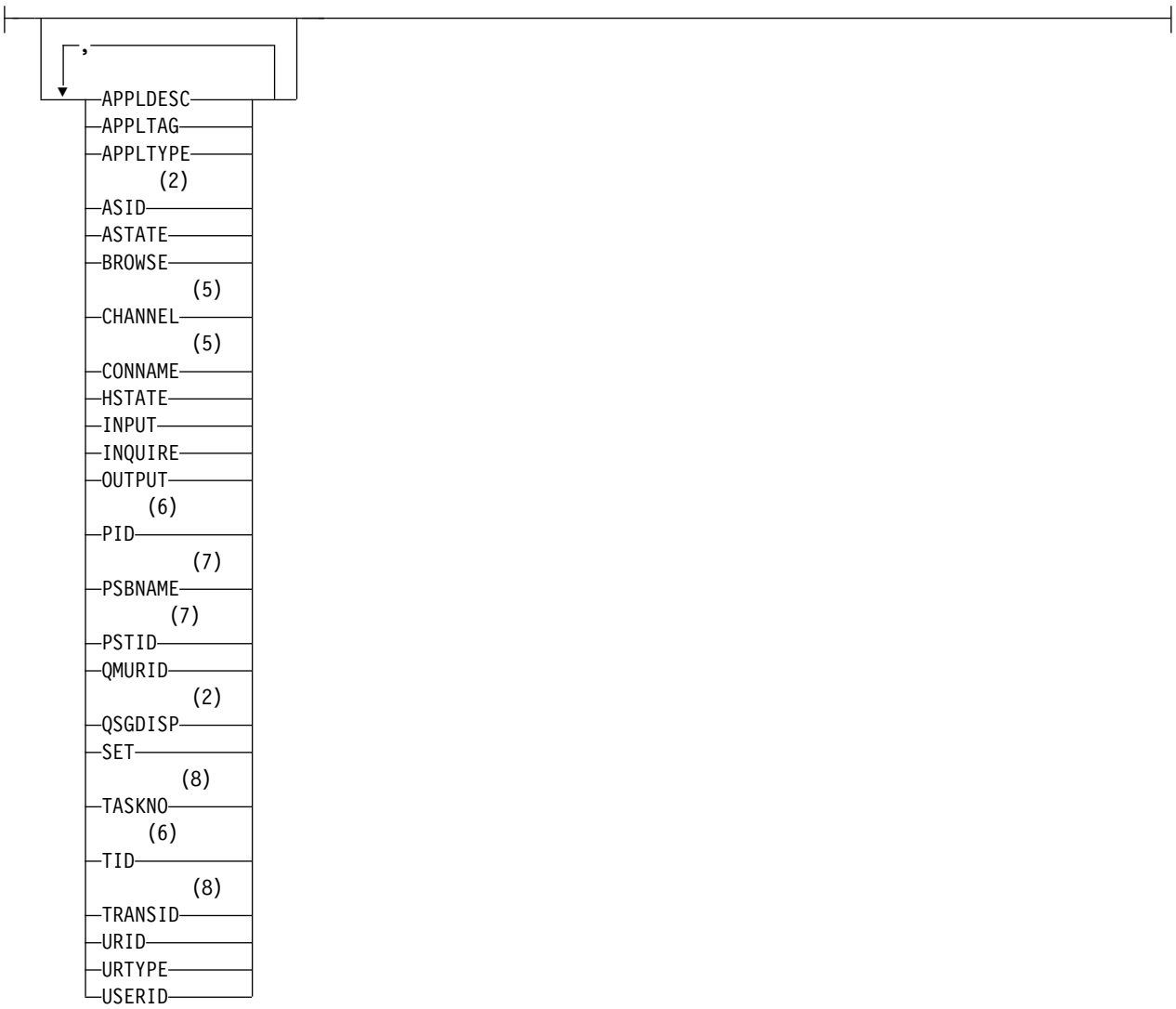
DISPLAY QSTATUS



Queue status:



Handle status:



Notes:

1 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

2 Valid on z/OS only.

3 Also displayed by selection of the MONITOR parameter.

4 Valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

5 Channel initiator only

6 Not valid on z/OS.

7 IMS only

8 CICS only

Usage notes for DISPLAY QSTATUS

The state of asynchronous consumers, ASTATE, reflects that of the server-connection proxy on behalf of the client application; it does not reflect the client application state.

Parameter descriptions for DISPLAY QSTATUS

You must specify the name of the queue for which you want to display status information. This name can either be a specific queue name or a generic queue name. By using a generic queue name you can display either:

- Status information for all queues, or
- Status information for one or more queues that match the specified name and other selection criteria

You must also specify whether you want status information about:

- Queues
- Handles that are accessing the queues

Note: You cannot use the DISPLAY QSTATUS command to display the status of an alias queue or remote queue. If you specify the name of one of these types of queue, no data is returned. You can, however, specify the name of the local queue or transmission queue to which the alias queue or remote queue resolves.

(*generic-qname***)**
The name of the queue for which status information is to be displayed. A trailing asterisk (*) matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own matches all queues.

WHERE
Specify a filter condition to display status information for queues that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE, MONITOR, OPENTYPE, QSGDISP, QTIME, TYPE, or URID parameters as filter keywords.

operator
The operator is used to determine whether a queue satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

CT Contains a specified item. If the *filter-keyword* is a list, you can use this filter to display objects whose attributes contain the specified item.

EX Does not contain a specified item. If the *filter-keyword* is a list, you can use this filter to display objects whose attributes do not contain the specified item.

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:
- An explicit value, that is a valid value for the attribute being tested.

You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value NO on the UNCOM parameter), you can only use EQ or NE.

- A generic value. This value is a character string (such as the character string in the APPLTAG parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.
- An item in a list of values. The operator must be CT or EX. If it is a character value, it can be explicit or generic. For example, if the value DEF is specified with the operator CT, all items where one of the attribute values is DEF are listed. If ABC* is specified, all items where one of the attribute values begins with ABC are listed.

ALL Display all the status information for each specified queue.

This value is the default if you do not specify a generic name, and do not request any specific parameters.

On z/OS, this value is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only the requested attributes are displayed.

CMDSCOPE

This parameter specifies how the command is executed when the queue manager is a member of a queue-sharing group. It is valid on z/OS only.

' ' The command is executed on the queue manager on which it was entered. This value is the default.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this value is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

MONITOR

Specify this value to return the set of online monitoring parameters. These are LGETDATE, LGETTIME, LPUTDATE, LPUTTIME, MONQ, MSGAGE, and QTIME. If you specify this parameter, any of the monitoring parameters that you request specifically have no effect; all monitoring parameters are still displayed.

OPENTYPE

Restricts the queues selected to queues which have handles with the specified type of access:

ALL Selects queues that are open with any type of access. This value is the default if the OPENTYPE parameter is not specified.

INPUT

Selects queues that are open for input only. This option does not select queues that are open for browse.

OUTPUT

Selects queues that are open only for output.

The OPENTYPE parameter is valid only if TYPE(HANDLE) is also specified.

You cannot use OPENTYPE as a filter keyword.

TYPE Specifies the type of status information required:

QUEUE
Status information relating to queues is displayed. This value is the default if the TYPE parameter is not specified.

HANDLE
Status information relating to the handles that are accessing the queues is displayed.

You cannot use TYPE as a filter keyword.

Queue status

For queue status, the following information is always returned for each queue that satisfies the selection criteria, except where indicated:

- Queue name
- Type of information returned (TYPE parameter)
- On platforms other than z/OS, current queue depth (CURDEPTH parameter)
- On z/OS only, the queue-sharing group disposition (QSGDISP parameter)

The following parameters can be specified for TYPE(QUEUE) to request additional information for each queue. If a parameter is specified that is not relevant for the queue, operating environment, or type of status information requested, that parameter is ignored.

CURDEPTH
The current depth of the queue, that is, the number of messages on the queue, including both committed messages and uncommitted messages.

IPPROCS
The number of handles that are currently open for input for the queue (either input-shared or input-exclusive). This number does not include handles that are open for browse.

For shared queues, the number returned applies only to the queue manager generating the reply. The number is not the total for all the queue managers in the queue-sharing group.

LGETDATE
The date on which the last message was retrieved from the queue since the queue manager started. A message being browsed does not count as a message being retrieved. When no get date is available, perhaps because no message has been retrieved from the queue since the queue manager was started, the value is shown as a blank. For queues with QSGDISP(SHARED), the value shown is for measurements collected on this queue manager only.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LGETTIME
The time at which the last message was retrieved from the queue since the queue manager started. A message being browsed does not count as a message being retrieved. When no get time is available, perhaps because no message has been retrieved from the queue since the queue manager was started, the value is shown as a blank. For queues with QSGDISP(SHARED), the value shown is for measurements collected on this queue manager only.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LPUTDATE
The date on which the last message was put to the queue since the queue manager started. When no put date is available, perhaps because no message has been put to the queue since the queue manager was started, the value is shown as a blank. For queues with QSGDISP(SHARED), the value shown is for measurements collected on this queue manager only.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

LPUTTIME
The time at which the last message was put to the queue since the queue manager started. When no put time is available, perhaps because no message has been put to the queue since the queue manager was started, the value is shown as a blank. For queues with QSGDISP(SHARED), the value shown is for measurements collected on this queue manager only.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

Note: Moving the system clock backwards should be avoided in case the LPUTTIME is being used to monitor the messages. The LPUTTIME of a queue is only updated when a message that arrives on the queue has a PutTime greater than the existing value of LPUTTIME. Because the PutTime of the message is less than the existing LPUTTIME of the queue in this case, the time is left unchanged.

MEDIALOG
The log extent or journal receiver needed for media recovery of the queue. On queue managers on which circular logging is in place, MEDIALOG is returned as a null string.

This parameter is valid on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

MONQ
Current level of monitoring data collection for the queue.

This parameter is also displayed when you specify the MONITOR parameter.

MSGAGE
Age, in seconds, of the oldest message on the queue. The maximum displayable value is 999999999; if the age exceeds this value, 999999999 is displayed.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

OPPROCS
This is the number of handles that are currently open for output for the queue.

For shared queues, the number returned applies only to the queue manager generating the reply. The number is not the total for all the queue managers in the queue-sharing group.

QSGDISP
Indicates the disposition of the queue. The value displayed is one of the following:

QMGR
The object was defined with QSGDISP(QMGR).

COPY The object was defined with QSGDISP(COPY).

SHARED
The object was defined with QSGDISP(SHARED).

This parameter is valid on z/OS only.

For shared queues, if the CF structure used by the queue is unavailable or has failed, the status information might be unreliable.

You cannot use QSGDISP as a filter keyword.

QTIME

Interval, in microseconds, between messages being put on the queue and then being destructively read. The maximum displayable value is 999999999; if the interval exceeds this value, 999999999 is displayed.

The interval is measured from the time that the message is placed on the queue until it is retrieved by an application and, therefore, includes any interval caused by a delay in committing by the putting application.

Two values are displayed:

- A value based on recent activity over a short time period.
- A value based on activity over a longer time period.

These values depend on the configuration and behavior of your system, as well as the levels of activity within it, and serve as an indicator that your system is performing normally. A significant variation in these values might indicate a problem with your system. For queues with QSGDISP(SHARED), the values shown are for measurements collected on this queue manager only.

This parameter is also displayed when you specify the MONITOR parameter.

A value is only displayed for this parameter if MONQ is set to a value other than OFF for this queue.

UNCOM

Indicates whether there are any uncommitted changes (puts and gets) pending for the queue. The value displayed is one of the following:

YES

On z/OS, there are one or more uncommitted changes pending.

NO There are no uncommitted changes pending.

n On platforms other than z/OS, an integer value indicating how many uncommitted changes are pending.

For shared queues, the value returned applies only to the queue manager generating the reply. The value does not apply to all the queue managers in the queue-sharing group.

Handle status

For handle status, the following information is always returned for each queue that satisfies the selection criteria, except where indicated:

- Queue name
- Type of information returned (TYPE parameter)
- On platforms other than z/OS, user identifier (USERID parameter) - not returned for APPLTYPE(SYSTEM)
- On platforms other than z/OS, process ID (PID parameter)
- On platforms other than z/OS, thread ID (TID parameter)
- On platforms other than z/OS, application tag (APPLTAG parameter)
- Application type (APPLTYPE parameter)
- On platforms other than z/OS, whether the handle provides input access (INPUT parameter)

- On platforms other than z/OS, whether the handle provides output access (OUTPUT parameter)
- On platforms other than z/OS, whether the handle provides browse access (BROWSE parameter)
- On platforms other than z/OS, whether the handle provides inquire access (INQUIRE parameter)
- On platforms other than z/OS, whether the handle provides set access (SET parameter)

The following parameters can be specified for TYPE(HANDLE) to request additional information for each queue. If a parameter that is not relevant is specified for the queue, operating environment, or type of status information requested, that parameter is ignored.

APPLDESC
A string containing a description of the application connected to the queue manager, where it is known. If the application is not recognized by the queue manager the description returned is blank.

APPLTAG
A string containing the tag of the application connected to the queue manager. It is one of the following:

- z/OS batch job name
- TSO USERID
- CICS APPLID
- IMS region name
- Channel initiator job name
- IBM i job name
- UNIX process

  Note: On HP-UX if the process name exceeds 14 characters, only the first 14 characters are shown. On all other platforms if the process name exceeds 28 characters, only the first 28 characters are shown.
- Windows process

  Note: The returned value consists of the full program path and executable file name. If it is more than 28 characters long, only the first 28 characters are shown.
- Internal queue manager process name

Application name represents the name of the process or job that has connected to the queue manager. In the instance that this process or job is connected via a channel, the application name represents the remote process or job rather than the local channel process or job name.

APPLTYPE
A string indicating the type of the application that is connected to the queue manager. It is one of the following:

BATCH
Application using a batch connection

RRSBATCH
RRS-coordinated application using a batch connection

CICS CICS transaction

IMS IMS transaction

CHINIT
Channel initiator

SYSTEM
Queue manager

SYSTEMEXT
Application performing an extension of function that is provided by the queue manager

USER A user application

ASID A four-character address-space identifier of the application identified by APPLTAG. It distinguishes duplicate values of APPLTAG.

This parameter is returned only when the queue manager owning the queue is running on z/OS, and the APPLTYPE parameter does not have the value SYSTEM.

ASTATE
The state of the asynchronous consumer on this queue.

Possible values are:

ACTIVE
An MQCB call has set up a function to call back to process messages asynchronously and the connection handle has been started so that asynchronous message consumption can proceed.

INACTIVE
An MQCB call has set up a function to call back to process messages asynchronously but the connection handle has not yet been started, or has been stopped or suspended, so that asynchronous message consumption cannot currently proceed.

SUSPENDED
The asynchronous consumption call-back has been suspended so that asynchronous message consumption cannot currently proceed on this queue. This can be either because an MQCB call with Operation MQOP_SUSPEND has been issued against this object handle by the application, or because it has been suspended by the system. If it has been suspended by the system, as part of the process of suspending asynchronous message consumption the call-back function is initiated with the reason code that describes the problem resulting in suspension. This code is reported in the Reason field in the MQCBC structure that is passed to the call-back function.

For asynchronous message consumption to proceed, the application must issue an MQCB call with the Operation parameter set to MQOP_RESUME.

SUSPTEMP
The asynchronous consumption call-back has been temporarily suspended by the system so that asynchronous message consumption cannot currently proceed on this queue. As part of the process of suspending asynchronous message consumption, the call-back function is called with the reason code that describes the problem resulting in suspension. This code is reported in the Reason field in the MQCBC structure passed to the call-back function.

The call-back function is initiated again when asynchronous message consumption is resumed by the system, when the temporary condition has been resolved.

NONE
An MQCB call has not been issued against this handle, so no asynchronous message consumption is configured on this handle.

BROWSE
Indicates whether the handle is providing browse access to the queue. The value is one of the following:

YES The handle is providing browse access.

NO The handle is not providing browse access.

CHANNEL
The name of the channel that owns the handle. If there is no channel associated with the handle, this parameter is blank.

This parameter is returned only when the handle belongs to the channel initiator.

CONNAME
The connection name associated with the channel that owns the handle. If there is no channel associated with the handle, this parameter is blank.

This parameter is returned only when the handle belongs to the channel initiator.

HSTATE
Whether an API call is in progress.

Possible values are:

ACTIVE
An API call from a connection is currently in progress for this object. For a queue, this condition can arise when an MQGET WAIT call is in progress.

If there is an MQGET SIGNAL outstanding, then this value does not mean, by itself, that the handle is active.

INACTIVE
No API call from a connection is currently in progress for this object. For a queue, this condition can arise when no MQGET WAIT call is in progress.

INPUT
Indicates whether the handle is providing input access to the queue. The value is one of the following:

SHARED
The handle is providing shared-input access.

EXCL The handle is providing exclusive-input access.

NO The handle is not providing input access.

INQUIRE
Indicates whether the handle currently provides inquire access to the queue. The value is one of the following:

YES The handle provides inquire access.

NO The handle does not provide inquire access.

OUTPUT
Indicates whether the handle is providing output access to the queue. The value is one of the following:

YES The handle is providing output access.

NO The handle is not providing output access.

PID Number specifying the process identifier of the application that has opened the specified queue.

This parameter is not valid on z/OS.

PSBNAME
The eight characters long name of the program specification block (PSB) associated with the running IMS transaction. You can use the PSBNAME and PSTID to purge the transaction using IMS commands. It is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value IMS.

PSTID
The four character IMS program specification table (PST) region identifier for the connected IMS region. It is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value IMS.

QMURID
The queue manager unit of recovery identifier. On z/OS, this value is a 6-byte log RBA, displayed as 12 hexadecimal characters. On platforms other than z/OS, this value is an 8-byte transaction identifier, displayed as `m.n` where `m` and `n` are the decimal representation of the first and last 4 bytes of the transaction identifier.

You can use QMURID as a filter keyword. On z/OS, you must specify the filter value as a hexadecimal string. On platforms other than z/OS, you must specify the filter value as a pair of decimal numbers separated by a period (.). You can only use the EQ, NE, GT, LT, GE, or LE filter operators.

QSGDISP
Indicates the disposition of the queue. It is valid on z/OS only. The value is one of the following:

QMGR
The object was defined with QSGDISP(QMGR).

COPY The object was defined with QSGDISP(COPY).

SHARED
The object was defined with QSGDISP(SHARED).

You cannot use QSGDISP as a filter keyword.

SET Indicates whether the handle is providing set access to the queue. The value is one of the following:

YES The handle is providing set access.

NO The handle is not providing set access.

TASKNO
A seven-digit CICS task number. This number can be used in the CICS command "CEMT SET TASK(taskno) PURGE" to end the CICS task. This parameter is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value CICS.

TID Number specifying the thread identifier within the application process that has opened the specified queue.

This parameter is not valid on z/OS.

An asterisk indicates that this queue was opened using a shared connection.

For further information about shared connections see Shared (thread independent) connections with MQCONNX.

TRANSID
A four-character CICS transaction identifier. This parameter is valid on z/OS only.

This parameter is returned only when the APPLTYPE parameter has the value CICS.

URID
The external unit of recovery identifier associated with the connection. It is the recovery identifier known in the external syncpoint coordinator. Its format is determined by the value of URTYPE.

You cannot use URID as a filter keyword.

URTYPE
The type of unit of recovery as seen by the queue manager. It is one of the following:
- CICS (valid only on z/OS)

- XA
- RRS (valid only on z/OS)
- IMS (valid only on z/OS)
- QMGR

URTYPE identifies the EXTURID type and not the type of the transaction coordinator. When URTYPE is QMGR, the associated identifier is in QMURID (and not URID).

USERID

The user identifier associated with the handle.

This parameter is not returned when APPLTYPE has the value SYSTEM.

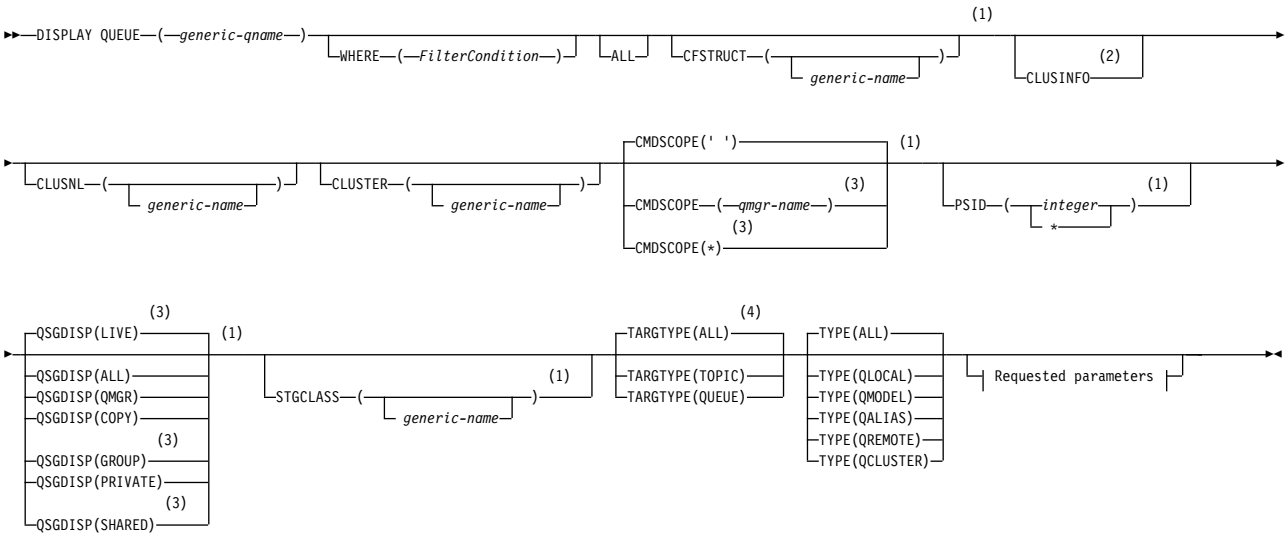
DISPLAY QUEUE:

Use the MQSC command **DISPLAY QUEUE** to display the attributes of one or more queues of any type.

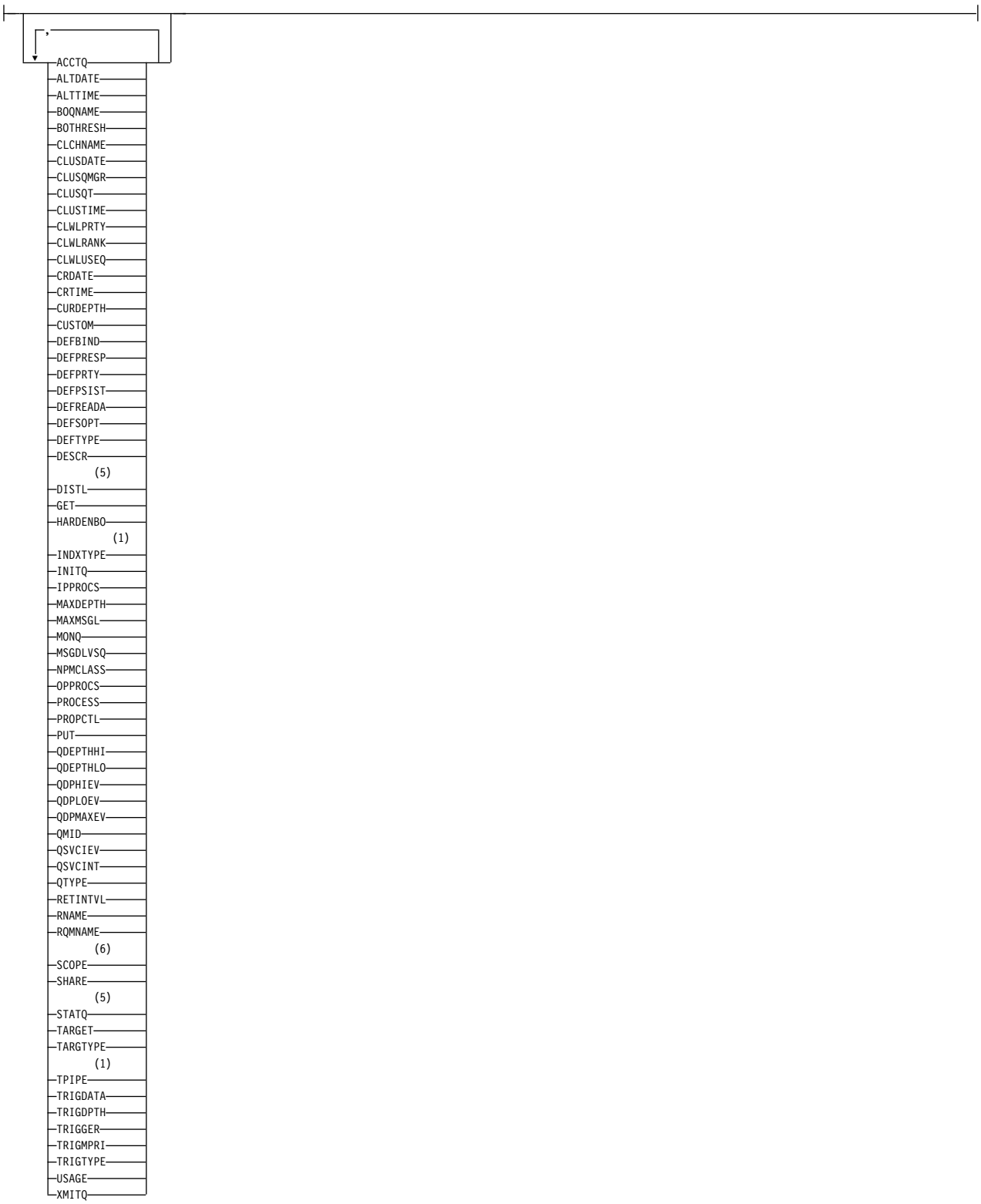
UNIX and Linux	Windows
✓	✓

Synonym: **DIS Q**

DISPLAY QUEUE



Requested parameters:



Notes:

1 Valid only on z/OS.

2 On z/OS, you cannot issue this from CSQINP2.

3 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

4 Valid only on an alias queue.

5 Not valid on z/OS.

6 Not valid on z/OS or IBM i.

Usage notes

1. You can use the following commands (or their synonyms) as an alternative way to display these attributes.
   - **DISPLAY QALIAS**
   - **DISPLAY QCLUSTER**
   - **DISPLAY QLOCAL**
   - **DISPLAY QMODEL**
   - **DISPLAY QREMOTE**

   These commands produce the same output as the DISPLAY QUEUE TYPE(*queue-type*) command. If you enter the commands this way, do not use the TYPE parameter.
2. On z/OS, the channel initiator must be running before you can display information about cluster queues (using TYPE(QCLUSTER) or the CLUSINFO parameter).
3. The command might not show every clustered queue in the cluster when issued on a partial repository, because the partial repository only knows about a queue once it has tried to use it.

Parameter descriptions for DISPLAY QUEUE

You must specify the name of the queue definition you want to display. This can be a specific queue name or a generic queue name. By using a generic queue name, you can display either:

- All queue definitions
- One or more queues that match the specified name

queue-name
The local name of the queue definition to be displayed (see Rules for naming IBM WebSphere MQ objects). A trailing asterisk * matches all queues with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all queues.

WHERE
Specify a filter condition to display only those queues that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this **DISPLAY** command. However, you cannot use the CMDSCOPE, QDPHIEV, QDPLOEV, QDPMAXEV, QSGDISP, or QSVCIEV parameters as filter keywords. You cannot use CFSTRUCT, CLUSTER, CLUSNL, PSID, or STGCLASS if these are also used to select queues. Queues of a type for which the filter keyword is not a valid attribute are not displayed.

operator
This is used to determine whether a queue satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value QALIAS on the CLUSQT parameter), you can only use EQ or NE. For the parameters HARDENBO, SHARE, and TRIGGER, use either EQ YES or EQ NO.
- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Specify this to display all the attributes. If this parameter is specified, any attributes that are also requested specifically have no effect; all attributes are still displayed.

On AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS, this is the default if you do not specify a generic name and do not request any specific attributes.

On z/OS this is also the default if you specify a filter condition using the `WHERE` parameter, but on other platforms only requested attributes are displayed.

CFSTRUCT(*generic-name)*
This parameter is optional and limits the information displayed to those queues where the value of the coupling facility structure is specified in brackets.

The value can be a generic name. If you do not enter a value for this parameter, **`CFSTRUCT`** is treated as a requested parameter.

CLUSINFO
This requests that, in addition to information about attributes of queues defined on this queue manager, information about these and other queues in the cluster that match the selection criteria is displayed. In this case, there might be multiple queues with the same name displayed. The cluster information is obtained from the repository on this queue manager.

This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS. Note that, on z/OS, you cannot issue `DISPLAY QUEUE CLUSINFO` commands from `CSQINP2`.

CLUSNL(*generic-name***)**
This is optional, and limits the information displayed if entered with a value in brackets:

- For queues defined on the local queue manager, only those with the specified cluster list. The value can be a generic name. Only queue types for which **`CLUSNL`** is a valid parameter are restricted in this way; other queue types that meet the other selection criteria are displayed.
- For cluster queues, only those belonging to clusters in the specified cluster list if the value is not a generic name. If the value is a generic name, no restriction is applied to cluster queues.

If you do not enter a value to qualify this parameter, it is treated as a requested parameter, and cluster list information is returned about all the queues displayed.

This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

Note: If the disposition requested is SHARED, CMDSCOPE must be blank or the local queue manager.

CLUSTER(*generic-name***)**
This is optional, and limits the information displayed to queues with the specified cluster name if entered with a value in brackets. The value can be a generic name. Only queue types for which **CLUSTER** is a valid parameter are restricted in this way by this parameter; other queue types that meet the other selection criteria are displayed.

If you do not enter a value to qualify this parameter, it is treated as a requested parameter, and cluster name information is returned about all the queues displayed.

This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP or SHARED.

" The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use **CMDSCOPE** as a filter keyword.

PSID(*integer***)**
The identifier of the page set where a queue resides. This is optional. Specifying a value limits the information displayed to queues that have an active association to the specified page set. The value consists of two numeric characters, in the range 00 - 99. An asterisk * on its own specifies all page set identifiers. If you do not enter a value, page set information is returned about all the queues displayed.

The page set identifier is displayed only if there is an active association of the queue to a page set, that is, after the queue has been the target of an MQPUT request. The association of a queue to a page set is not active when:

- The queue is just defined
- The STGCLASS attribute of the queue is altered, and there is no subsequent MQPUT request to the queue
- The queue manager is restarted and there are no messages on the queue

This parameter is valid only on z/OS.

QSGDISP
Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE This is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, also display information for objects defined with QSGDISP(SHARED).

ALL Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP) or QSGDISP(SHARED).

In a shared queue manager environment:

```
DISPLAY QUEUE(name) CMDSCOPE(*) QSGDISP(ALL)
```

The command lists objects matching name in the queue-sharing group, without duplicating those in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information only for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

SHARED
Display information only for objects defined with QSGDISP(SHARED). This is allowed only in a shared queue manager environment.

Note: For cluster queues, this is always treated as a requested parameter. The value returned is the disposition of the real queue that the cluster queue represents.

If QSGDISP(LIVE) is specified or defaulted, or if QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions) .

Note: In the QSGDISP(LIVE) case, this occurs only where a shared and a non-shared queue have the same name; such a situation should not occur in a well-managed system.

QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

SHARED
The object was defined with QSGDISP(SHARED).

You cannot use **QSGDISP** as a filter keyword.

STGCLASS(*generic-name)*
This is optional, and limits the information displayed to queues with the storage class specified if entered with a value in brackets. The value can be a generic name.

If you do not enter a value to qualify this parameter, it is treated as a requested parameter, and storage class information is returned about all the queues displayed.

This parameter is valid only on z/OS.

TARGTYPE(*target-type***)**
This is optional and specifies the target type of the alias queue you want to be displayed.

TYPE(*queue-type***)**

This is optional, and specifies the type of queues you want to be displayed. If you specify ALL, which is the default value, all queue types are displayed; this includes cluster queues if CLUSINFO is also specified.

As well as ALL, you can specify any of the queue types allowed for a **DEFINE** command: QALIAS, QLOCAL, QMODEL, QREMOTE, or their synonyms, as follows:

QALIAS
Alias queues

QLOCAL
Local queues

QMODEL
Model queues

QREMOTE
Remote queues

You can specify a queue type of QCLUSTER to display only cluster queue information. If QCLUSTER is specified, any selection criteria specified by the CFSTRUCT, STGCLASS, or PSID parameters are ignored. Note that you cannot issue **DISPLAY QUEUE TYPE(QCLUSTER)** commands from CSQINP2.

On platforms other than z/OS, **QTYPE**(*type*)can be used as a synonym for this parameter.

The queue name and queue type (and, on z/OS, the queue disposition) are always displayed.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

Most parameters are relevant only for queues of a particular type or types. Parameters that are not relevant for a particular type of queue cause no output, nor is an error raised.

The following table shows the parameters that are relevant for each type of queue. There is a brief description of each parameter after the table, but for more information, see the **DEFINE** command for each queue type.

Table 86. Parameters that can be returned by the **DISPLAY QUEUE** *command.*

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
ACCTQ	✓	✓			
ALTDATE	✓	✓	✓	✓	✓
ALTTIME	✓	✓	✓	✓	✓
BOQNAME	✓	✓			
BOTHRESH	✓	✓			
CFSTRUCT	✓	✓			

Table 86. Parameters that can be returned by the **DISPLAY QUEUE** *command (continued).*

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
CLCHNAME	✓	✓			
CLUSDATE					✓
CLUSNL	✓		✓	✓	
CLUSQMGR					✓
CLUSQT					✓
CLUSTER	✓		✓	✓	✓
CLUSTIME					✓
CLWLPRTY	✓		✓	✓	✓
CLWLRANK	✓		✓	✓	✓
CLWLUSEQ	✓				
CRDATE	✓	✓			
CRTIME	✓	✓			
CURDEPTH	✓				
CUSTOM	✓	✓	✓	✓	✓
DEFBIND	✓		✓	✓	✓
DEFPRESP	✓	✓	✓	✓	✓
DEFPRTY	✓	✓	✓	✓	✓
DEFPSIST	✓	✓	✓	✓	✓
DEFREADA	✓	✓	✓		
DEFSOPT	✓	✓			
DEFTYPE	✓	✓			
DESCR	✓	✓	✓	✓	✓
DISTL	✓	✓			

Table 86. Parameters that can be returned by the **DISPLAY QUEUE** *command (continued).*

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
GET	✓	✓	✓		
HARDENBO	✓	✓			
INDXTYPE	✓	✓			
INITQ	✓	✓			
IPPROCS	✓				
MAXDEPTH	✓	✓			
MAXMSGL	✓	✓			
MONQ	✓	✓			
MSGDLVSQ	✓	✓			
NPMCLASS	✓	✓			
OPPROCS	✓				
PROCESS	✓	✓			
PROPCTL	✓	✓	✓		
PSID	✓				
PUT	✓	✓	✓	✓	✓
QDEPTHHI	✓	✓			
QDEPTHLO	✓	✓			
QDPHIEV	✓	✓			
QDPLOEV	✓	✓			
QDPMAXEV	✓	✓			
QMID					✓
QSGDISP	✓	✓	✓	✓	✓
QSVCIEV	✓	✓			

Table 86. Parameters that can be returned by the **DISPLAY QUEUE** *command (continued).*

Cross-tabulation of queue parameters and queue types. If the parameter applies to the queue type, the cell contains a check mark.

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
QSVCINT	✓	✓			
QTYPE	✓	✓	✓	✓	✓
RETINTVL	✓	✓			
RNAME				✓	
RQMNAME				✓	
SCOPE	✓		✓	✓	
SHARE	✓	✓			
STATQ	✓	✓			
STGCLASS	✓	✓			
TARGET			✓		
TARGTYPE			✓		
TPIPE	✓				
TRIGDATA	✓	✓			
TRIGDPTH	✓	✓			
TRIGGER	✓	✓			
TRIGMPRI	✓	✓			
TRIGTYPE	✓	✓			
USAGE	✓	✓			
XMITQ				✓	

ACCTQ
Whether accounting (on z/OS, thread-level and queue-level accounting) data collection is to be enabled for the queue.

ALTDATE
The date on which the definition or information was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition or information was last altered, in the form `hh.mm.ss`.

BOQNAME
Backout requeue name.

BOTHRESH
Backout threshold.

CLCHNAME
CLCHNAME is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue. CLCHNAME is not supported on z/OS.

CLUSDATE
The date on which the definition became available to the local queue manager, in the form yyyy-mm-dd.

CLUSNL
The namelist that defines the cluster that the queue is in.

CLUSQMGR
The name of the queue manager that hosts the queue.

CLUSQT
Cluster queue type. This can be:

QALIAS
The cluster queue represents an alias queue.

QLOCAL
The cluster queue represents a local queue.

QMGR
The cluster queue represents a queue manager alias.

QREMOTE
The cluster queue represents a remote queue.

CLUSTER
The name of the cluster that the queue is in.

CLUSTIME
The time at which the definition became available to the local queue manager, in the form hh.mm.ss.

CLWLPRTY
The priority of the queue for the purposes of cluster workload distribution.

CLWLRANK
The rank of the queue for the purposes of cluster workload distribution.

CLWLUSEQ
Whether puts are allowed to other queue definitions apart from local ones.

CRDATE
The date on which the queue was defined (in the form yyyy-mm-dd).

CRTIME
The time at which the queue was defined (in the form hh.mm.ss).

CURDEPTH
Current depth of queue.

On z/OS, CURDEPTH is returned as zero for queues defined with a disposition of GROUP. It is also returned as zero for queues defined with a disposition of SHARED if the CF structure that they use is unavailable or has failed.

Messages put on a queue count toward the current depth as they are put. Messages got from a queue do not count toward the current depth. This is true whether operations are done under syncpoint or not. Commit has no effect on current depth. Therefore:

- Messages put under syncpoint (but not yet committed) are included in the current depth.
- Messages got under syncpoint (but not yet committed) are not included in the current depth.

CUSTOM
This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value in the form `NAME(VALUE)`.

DEFBIND
Default message binding.

DEFPRESP
Default put response; defines the behavior that should be used by applications when the put response type in the MQPMO options has been set to MQPMO_RESPONSE_AS_Q_DEF.

DEFPRTY
Default priority of the messages put on the queue.

DEFPSIST
Whether the default persistence of messages put on this queue is set to NO or YES. NO means that messages are lost across a restart of the queue manager.

DEFREADA
This specifies the default read ahead behavior for non-persistent messages delivered to the client.

DEFSOPT
Default share option on a queue opened for input.

DEFTYPE
Queue definition type. This can be:

- PREDEFINED (Predefined)

  The queue was created with a DEFINE command, either by an operator or by a suitably authorized application sending a command message to the service queue.
- PERMDYN (Permanent dynamic)

  Either the queue was created by an application issuing **MQOPEN** with the name of a model queue specified in the object descriptor (MQOD), or (if this is a model queue) this determines the type of dynamic queue that can be created from it.

  On z/OS the queue was created with `QSGDISP(QMGR)`.
- TEMPDYN (Temporary dynamic)

  Either the queue was created by an application issuing **MQOPEN** with the name of a model queue specified in the object descriptor (MQOD), or (if this is a model queue) this determines the type of dynamic queue that can be created from it.

  On z/OS the queue was created with `QSGDISP(QMGR)`.
- SHAREDYN

  A permanent dynamic queue was created when an application issued an **MQOPEN** API call with the name of this model queue specified in the object descriptor (MQOD).

  On z/OS, in a queue-sharing group environment, the queue was created with `QSGDISP(SHARED)`.

DESCR
Descriptive comment.

DISTL
Whether distribution lists are supported by the partner queue manager. (Supported only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.)

GET Whether the queue is enabled for gets.

HARDENBO
Whether the back out count is hardened to ensure that the count of the number of times that a message has been backed out is accurate.

Note: This parameter affects only WebSphere MQ for z/OS. It can be set and displayed on other platforms but has no effect.

INDXTYPE
Index type (supported only on z/OS).

INITQ
Initiation queue name.

IPPROCS
Number of handles indicating that the queue is open for input.

On z/OS, IPPROCS is returned as zero for queues defined with a disposition of GROUP. With a disposition of SHARED, only the handles for the queue manager sending back the information are returned, not the information for the whole group.

MAXDEPTH
Maximum depth of queue.

MAXMSGL
Maximum message length.

MONQ
Online monitoring data collection.

MSGDLVSQ
Message delivery sequence.

NPMCLASS
Level of reliability assigned to non-persistent messages that are put to the queue.

OPPROCS
Number of handles indicating that the queue is open for output.

On z/OS, OPPROCS is returned as zero for queues defined with a disposition of GROUP. With a disposition of SHARED, only the handles for the queue manager sending back the information are returned, not the information for the whole group.

PROCESS
Process name.

PROPCTL
Property control attribute.

This parameter is applicable to Local, Alias and Model queues.

This parameter is optional.

Specifies how message properties are handled when messages are retrieved from queues using the MQGET call with the `MQGMO_PROPERTIES_AS_Q_DEF` option.

Permissible values are:

ALL To contain all the properties of the message, except those contained in the message descriptor (or extension), select `All`. The `All` value enables applications that cannot be changed to access all the message properties from `MQRFH2` headers.

COMPAT
If the message contains a property with a prefix of **`mcd.`**, **`jms.`**, **`usr.`**, or **`mqext.`**, all message properties are delivered to the application in an `MQRFH2` header. Otherwise all

properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

FORCE
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible via the message handle.

NONE
All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is delivered to the application.

PUT Whether the queue is enabled for puts.

QDEPTHHI
Queue Depth High event generation threshold.

QDEPTHLO
Queue Depth Low event generation threshold.

QDPHIEV
Whether Queue Depth High events are generated.

You cannot use QDPHIEV as a filter keyword.

QDPLOEV
Whether Queue Depth Low events are generated.

You cannot use QDPLOEV as a filter keyword.

QDPMAXEV
Whether Queue Full events are generated.

You cannot use QDPMAXEV as a filter keyword.

QMID
The internally generated unique name of the queue manager that hosts the queue.

QSVCIEV
Whether service interval events are generated.

You cannot use QSVCIEV as a filter keyword.

QSVCINT
Service interval event generation threshold.

QTYPE
Queue type.

On AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS, the queue type is always displayed.

On AIX, HP-UX, Linux, IBM i, Solaris, and Windows, TYPE(*type*) can be used as a synonym for this parameter.

RETINTVL
Retention interval.

RNAME
Name of the local queue, as known by the remote queue manager.

RQMNAME
Remote queue manager name.

SCOPE
Scope of queue definition (not supported on z/OS).

SHARE
Whether the queue can be shared.

STATQ
Whether statistics data information is to be collected.

STGCLASS
Storage class.

TARGET
This parameter requests that the base object name of an aliased queue is displayed.

TARGTYPE
This parameter requests that the target (base) type of an aliased queue is displayed.

TPIPE The TPIPE names used for communication with OTMA via the WebSphere MQ IMS bridge if the bridge is active. This parameter is supported only on z/OS.

TRIGDATA
Trigger data.

TRIGDPTH
Trigger depth.

TRIGGER
Whether triggers are active.

TRIGMPRI
Threshold message priority for triggers.

TRIGTYPE
Trigger type.

USAGE
Whether the queue is a transmission queue.

XMITQ
Transmission queue name.

For more details of these parameters, see "DEFINE queues" on page 511.

DISPLAY SBSTATUS:

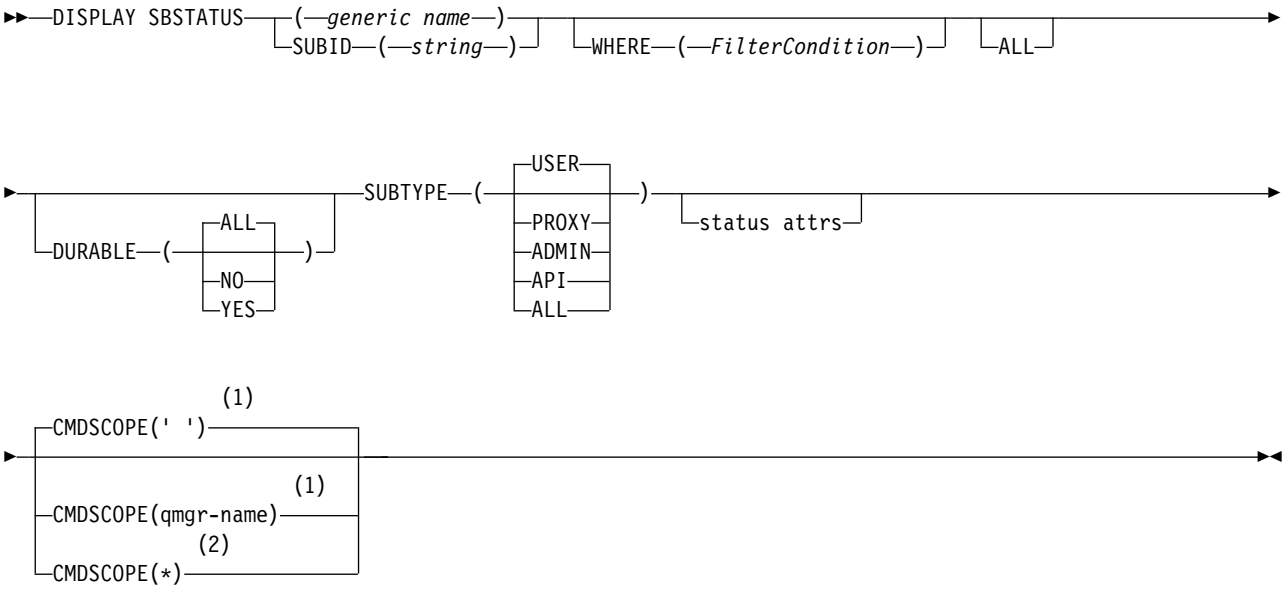
Use the MQSC command DISPLAY SBSTATUS to display the status of a subscription.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for DISPLAY SBSTATUS" on page 714
- "Requested parameters" on page 716

Synonym: DIS SBSTATUS

DISPLAY SBSTATUS



Status attributes:

Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Not valid on z/OS.

Parameter descriptions for DISPLAY SBSTATUS

You must specify the name of the subscription definition for which you want to display status information. This can be a specific subscription name or a generic subscription name. By using a generic subscription name, you can display either:

- All subscription definitions
- One or more subscriptions that match the specified name

(generic-name)
The local name of the subscription definition to be displayed. A trailing asterisk (*) matches all subscriptions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all subscriptions.

WHERE

Specify a filter condition to display only those subscriptions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword

Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE parameter as a filter keyword. Subscriptions of a type for which the filter keyword is not a valid attribute are not displayed.

operator

This is used to determine whether a subscription satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value

The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value USER on the SUBTYPE parameter), you can only use EQ or NE.
- A generic value. This is a character string (such as the character string you supply for the SUBUSER parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Display all the status information for each specified subscription definition. This is the default if you do not specify a generic name, and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only, requested attributes are displayed.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is processed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is processed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

DURABLE
Specify this attribute to restrict the type of subscriptions which are displayed.

ALL Display all subscriptions.

NO Only information about nondurable subscriptions is displayed.

YES Only information about durable subscriptions is displayed.

SUBTYPE
Specify this attribute to restrict the type of subscriptions which are displayed.

USER Displays only **API** and **ADMIN** subscriptions.

PROXY
Only system created subscriptions relating to inter-queue-manager subscriptions are selected.

ADMIN
Only subscriptions that have been created by an administration interface or modified by an administration interface are selected.

API Only subscriptions created by applications using a WebSphere MQ API call are selected.

ALL All subscription types are displayed (no restriction).

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

ACTCONN
Returns the *ConnId* of the *HConn* that currently has this subscription open.

DURABLE
A durable subscription is not deleted when the creating application closes its subscription handle.

NO The subscription is removed when the application that created it is closed or disconnected from the queue manager.

YES The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.

LMSGDATE
The date on which a message was last published to the destination specified by this subscription.

LMSGTIME
The time on which a message was last published to the destination specified by this subscription.

MCASTREL
Indicator of the reliability of the multicast messages.

The values are expressed as a percentage. A value of 100 indicates that all messages are being delivered without problems. A value less than 100 indicates that some of the messages are

experiencing network issues. To determine the nature of these issues the user can switch on event message generation, using the **COMMEV** parameter of the COMMINFO objects, and examine the generated event messages.

The following two values are returned:

- The first value is based on recent activity over a short period.
- The second value is based on activity over a longer period.

If no measurement is available the values are shown as blanks.

NUMMSGS
The number of messages put to the destination specified by this subscription since it was created, or since the queue manager was restarted, whichever is more recent. This number might not reflect the total number of messages that are, or have been, available to the consuming application. This is because it might also include publications that were partially processed but then undone by the queue manager due to a publication failure, or publications that were made within syncpoint that were rolled-back by the publishing application.

RESMDATE
The date of the most recent **MQSUB** API call that connected to the subscription.

RESMTIME
The time of the most recent **MQSUB** API call that connected to the subscription.

SUBID(*string***)**
The internal, unique key identifying a subscription.

SUBTYPE
Indicates how the subscription was created.

PROXY
An internally created subscription used for routing publications through a queue manager.

ADMIN
Created using the **DEF SUB** MQSC or PCF command. This **SUBTYPE** also indicates that a subscription has been modified using an administrative command.

API Created using an **MQSUB** API call.

For more details of these parameters, see "DEFINE SUB" on page 547

DISPLAY SERVICE:

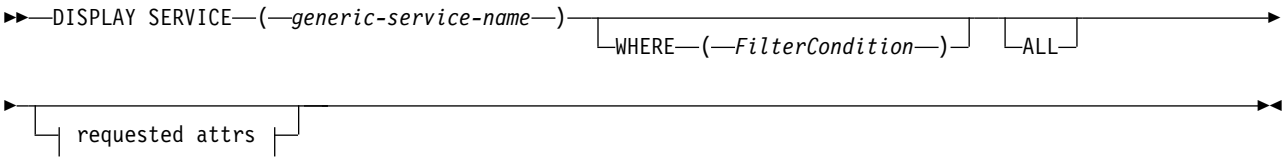
Use the MQSC command DISPLAY SERVICE to display information about a service.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Keyword and parameter descriptions for DISPLAY SERVICE" on page 718
- "Requested parameters" on page 719

Synonym:

DISPLAY SERVICE



Requested attrs:

Keyword and parameter descriptions for DISPLAY SERVICE

You must specify a service for which you want to display information. You can specify a service by using either a specific service name or a generic service name. By using a generic service name, you can display either:

- Information about all service definitions, by using a single asterisk (*), or
- Information about one or more service that match the specified name.

(*generic-service-name***)**
The name of the service definition for which information is to be displayed. A single asterisk (*) specifies that information for all service identifiers is to be displayed. A character string with an asterisk at the end matches all services with the string followed by zero or more characters.

WHERE
Specify a filter condition to display information for those listeners that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a listener satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value MANUAL on the CONTROL parameter), you can only use EQ or NE..
- A generic value. This is a character string. with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Specify this to display all the service information for each specified service. If this parameter is specified, any parameters that are requested specifically have no effect; all parameters are still displayed.

This is the default if you do not specify a generic identifier, and do not request any specific parameters.

On z/OS this is also the default if you specify a filter condition using the WHERE parameter, but on other platforms only requested attributes are displayed.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

ALTDATE
The date on which the definition was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition was last altered, in the form `hh.mm.ss`.

CONTROL
How the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the START SERVICE and STOP SERVICE commands.

QMGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR
Descriptive comment.

SERVTYPE
Specifies the mode in which the service is to run:

COMMAND
A command service object. Multiple instances of a command service object can be executed concurrently. You cannot monitor the status of command service objects.

SERVER
A server service object. Only one instance of a server service object can be executed at a time. The status of server service objects can be monitored using the DISPLAY SVSTATUS command.

STARTARG
Specifies the arguments to be passed to the user program at queue manager startup.

STARTCMD
Specifies the name of the program which is to run.

STDERR
Specifies the path to the file to which the standard error (stderr) of the service program is to be redirected.

STDOUT
Specifies the path to the file to which the standard output (stdout) of the service program is to be redirected.

STOPARG
Specifies the arguments to be passed to the stop program when instructed to stop the service.

STOPCMD
Specifies the name of the executable program to run when the service is requested to stop.

For more details of these parameters, see "DEFINE SERVICE" on page 544.

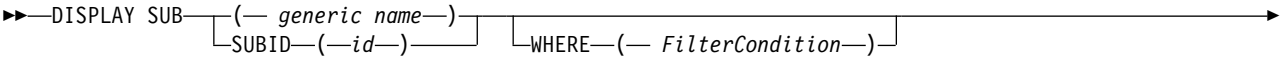
DISPLAY SUB:

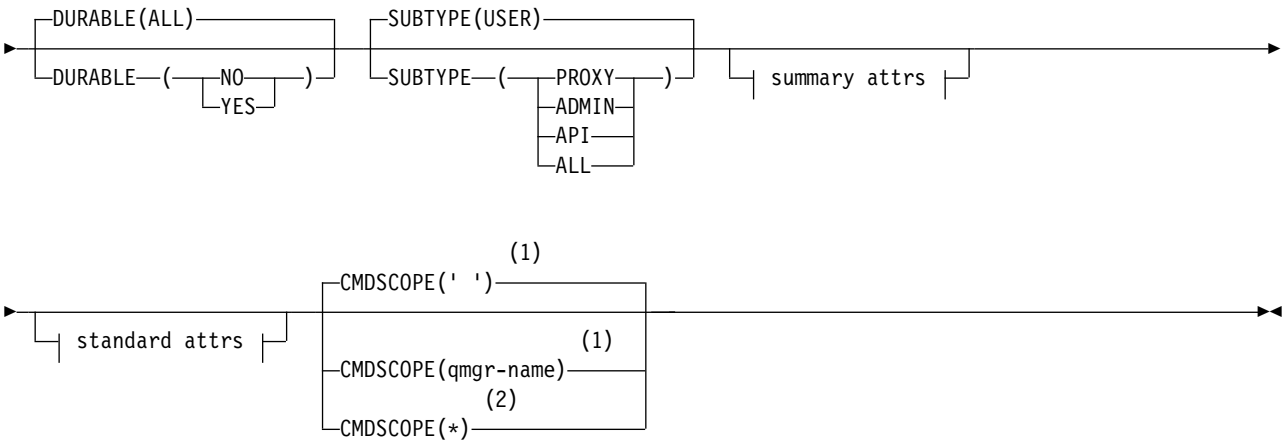
Use the MQSC command DISPLAY SUB to display the attributes associated with a subscription.

UNIX and Linux	Windows
✓	✓

Synonym: DIS SUB

DISPLAY SUB





summary attributes:

standard attributes:

Notes:

1 Valid only on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

Usage notes for DISPLAY SUB

1. The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed. On z/OS, these non-printable characters will be displayed as blanks. On distributed platforms using runmqsc, these non-printable characters will be displayed as dots.

Parameter descriptions for DISPLAY SUB

You must specify either the name or the identifier of subscription you want to display. This can be a specific subscription name, or SUBID, or a generic subscription name. By using a generic subscription name, you can display either:

- All subscription definitions
- One or more subscriptions that match the specified name

The following are valid forms:

```
DIS SUB(xyz)
DIS SUB SUBID(123)
DIS SUB(xyz*)
```

(generic-name)
The local name of the subscription definition to be displayed. A trailing asterisk (*) matches all subscriptions with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all subscriptions.

WHERE
Specify a filter condition to display only those subscriptions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE parameter as a filter keyword. Subscriptions of a type for which the filter keyword is not a valid attribute are not displayed.

operator
This is used to determine whether a subscription satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value QALIAS on the

CLUSQT parameter), you can only use EQ or NE. For the parameters HARDENBO, SHARE, and TRIGGER, use either EQ YES or EQ NO.

- A generic value. This is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

SUMMARY
Specify this to display the set of summary attributes; this is the default value.

On AIX, HP-UX, Linux, IBM i, Solaris, Windows , and z/OS, this is the default if you do not specify a generic name and do not request any specific attributes.

ALL Specify this to display all the attributes.

If this parameter is specified, any attributes that are also requested specifically have no effect; all attributes are still displayed.

ALTDATE(*string***)**
The date of the most recent **MQSUB** or **ALTER SUB** command that modified the properties of the subscription.

ALTTIME(*string***)**
The time of the most recent **MQSUB** or **ALTER SUB** command that modified the properties of the subscription.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is processed when the queue manager is a member of a queue-sharing group.

' ' The command is processed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is processed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of setting this value is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

CRDATE(*string***)**
The date of the first **MQSUB** or **DEF SUB** command that created this subscription.

CRTIME(*string***)**
The time of the first **MQSUB** or **DEF SUB** command that created this subscription.

DEST(*string***)**
The destination for messages published to this subscription; this parameter is the name of a queue.

DESTCLAS
System managed destination.

PROVIDED
The destination is a queue.

MANAGED
The destination is managed.

DESTCORL(*string***)**
The *CorrelId* used for messages published to this subscription.

DESTQMGR(*string***)**
The destination queue manager for messages published to this subscription.

DURABLE
A durable subscription is not deleted when the creating application closes its subscription handle.

ALL Display all subscriptions.

NO The subscription is removed when the application that created it, is closed or disconnected from the queue manager.

YES The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.

EXPIRY
The time to expiry of the subscription object from the creation date and time.

(*integer***)**
The time to expiry, in tenths of a second, from the creation date and time.

UNLIMITED
There is no expiry time. This is the default option supplied with the product.

PSPROP
The manner in which publish subscribe related message properties are added to messages sent to this subscription.

NONE
Do not add publish subscribe properties to the message.

COMPAT
Publish subscribe properties are added within an MQRFH version 1 header unless the message was published in PCF format.

MSGPROP
Publish subscribe properties are added as message properties.

RFH2 Publish subscribe properties are added within an MQRFH version 2 header.

PUBACCT(*string***)**
Accounting token passed by the subscriber, for propagation into messages published to this subscription in the *AccountingToken* field of the MQMD.

PUBAPPID(*string***)**
Identity data passed by the subscriber, for propagation into messages published to this subscription in the *ApplIdentityData* field of the MQMD.

PUBPRTY
The priority of the message sent to this subscription.

ASPUB
Priority of the message sent to this subscription is taken from the priority supplied in the published message.

ASQDEF
Priority of the message sent to this subscription is taken from the default priority of the queue defined as a destination.

(integer)
An integer providing an explicit priority for messages published to this subscription.

REQONLY
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription.

NO All publications on the topic are delivered to this subscription.

YES Publications are only delivered to this subscription in response to an MQSUBRQ API call.

This parameter is equivalent to the subscribe option MQSO_PUBLICATIONS_ON_REQUEST.

SELECTOR(*string***)**
A selector that is applied to messages published to the topic.

SELTYPE
The type of selector string that has been specified.

NONE
No selector has been specified.

STANDARD
The selector references only the properties of the message, not its content, using the standard WebSphere MQ selector syntax. Selectors of this type are to be handled internally by the queue manager.

EXTENDED
The selector uses extended selector syntax, typically referencing the content of the message. Selectors of this type cannot be handled internally by the queue manager; extended selectors can be handled only by another program such as WebSphere Message Broker.

SUB(*string***)**
The application's unique identifier for a subscription.

SUBID(*string***)**
The internal, unique key identifying a subscription.

SUBLEVEL(*integer***)**
The level within the subscription hierarchy at which this subscription is made. The range is zero through 9.

SUBSCOPE
Determines whether this subscription is forwarded to other queue managers, so that the subscriber receives messages published at those other queue managers.

ALL The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy.

QMGR
The subscription forwards messages published on the topic only within this queue manager.

Note: Individual subscribers can only *restrict* **SUBSCOPE**. If the parameter is set to ALL at topic level, then an individual subscriber can restrict it to QMGR for this subscription. However, if the parameter is set to QMGR at topic level, then setting an individual subscriber to ALL has no effect.

SUBTYPE
Indicates how the subscription was created.

USER Displays only **API** and **ADMIN** subscriptions.

PROXY
An internally created subscription used for routing publications through a queue manager.

ADMIN
Created using **DEF SUB** MQSC or PCF command. This **SUBTYPE** also indicates that a subscription has been modified using an administrative command.

API Created using an **MQSUB** API request.

ALL All.

SUBUSER(*string***)**
Specifies the user ID that is used for security checks that are performed to ensure that publications can be put to the destination queue associated with the subscription. This ID is either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription. The length of this parameter must not exceed 12 characters.

TOPICOBJ(*string***)**
The name of a topic object used by this subscription.

TOPICSTR(*string***)**
Specifies a fully qualified topic name, or a topic set using wildcard characters for the subscription.

USERDATA(*string***)**
Specifies the user data associated with the subscription. The string is a variable length value that can be retrieved by the application on an MQSUB API call and passed in a message sent to this subscription as a message property.

V 7.5.0.8 From Version 7.5.0, Fix Pack 8, an IBM WebSphere MQ classes for JMS application can retrieve the subscription user data from the message by using the constant JMS_IBM_SUBSCRIPTION_USER_DATA in the JmsConstants interface with the method javax.jms.Message.getStringProperty(java.lang.String). For more information, see Retrieval of user subscription data.

VARUSER
Specifies whether a user other than the subscription creator can connect to and take over ownership of the subscription.

ANY Any user can connect to and takeover ownership of the subscription.

FIXED
Takeover by another **USERID** is not permitted.

WSCHEMA
The schema to be used when interpreting any wildcard characters in the topic string.

CHAR
Wildcard characters represent portions of strings.

TOPIC
Wildcard characters represent portions of the topic hierarchy.

DISPLAY SVSTATUS:

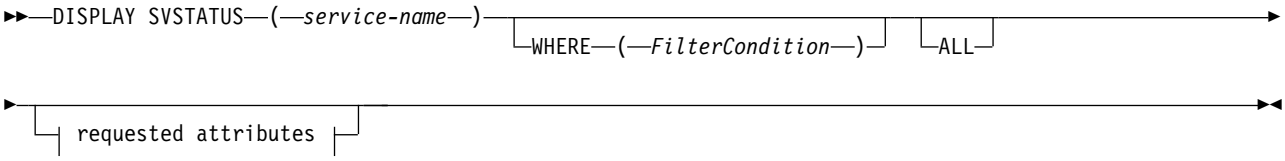
Use the MQSC command DISPLAY SVSTATUS to display status information for one or more services. Only services with a **SERVTYPE** of SERVER are displayed.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Keyword and parameter descriptions for DISPLAY SVSTATUS"
- "Requested parameters" on page 728

Synonym:

DISPLAY SVSTATUS



Requested attributes:

Keyword and parameter descriptions for DISPLAY SVSTATUS

You must specify a service for which you want to display status information. You can specify a service by using either a specific service name or a generic service name. By using a generic service name, you can display either:

- Status information for all service definitions, by using a single asterisk (*), or
- Status information for one or more services that match the specified name.

(*generic-service-name***)**
The name of the service definition for which status information is to be displayed. A single asterisk (*) specifies that information for all connection identifiers is to be displayed. A character string with an asterisk at the end matches all services with the string followed by zero or more characters.

WHERE

Specify a filter condition to display status information for those services that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Any parameter that can be used to display attributes for this DISPLAY command.

operator
This is used to determine whether a service satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter (for example, the value MANUAL on the CONTROL parameter), you can only use EQ or NE.
- A generic value. This is a character string. with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Display all the status information for each specified service. This is the default if you do not specify a generic name, and do not request any specific parameters.

Requested parameters

Specify one or more attributes that define the data to be displayed. The attributes can be specified in any order. Do not specify the same attribute more than once.

CONTROL

How the service is to be started and stopped:

MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by use of the START SERVICE and STOP SERVICE commands.

QMGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

STARTONLY
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

DESCR

Descriptive comment.

PID The operating system process identifier associated with the service.

SERVTYPE
The mode in which the service runs. A service can have a **SERVTYPE** of SERVER or COMMAND, but only services with **SERVTYPE(SERVER)** are displayed by this command.

STARTARG
The arguments passed to the user program at startup.

STARTCMD
The name of the program being run.

STARTDA
The date on which the service was started.

STARTTI
The time at which the service was started.

STATUS
The status of the process:

RUNNING
The service is running.

STARTING
The service is in the process of initializing.

STOPPING
The service is stopping.

STDERR
Destination of the standard error (stderr) of the service program.

STDOUT
Destination of the standard output (stdout) of the service program.

STOPARG
The arguments to be passed to the stop program when instructed to stop the service.

STOPCMD
The name of the executable program to run when the service is requested to stop.

For more details of these parameters, see "DEFINE SERVICE" on page 544.

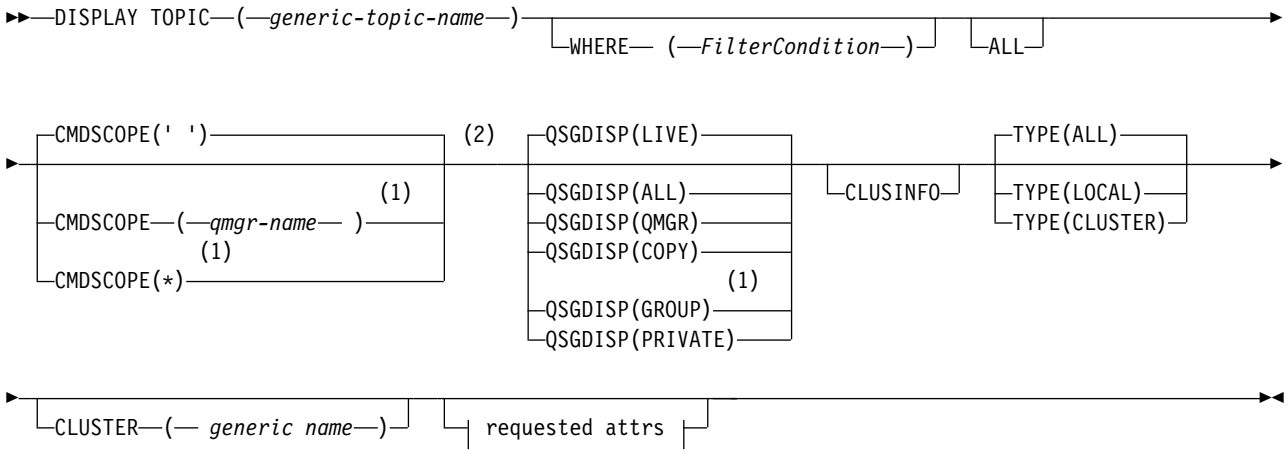
DISPLAY TOPIC:

Use the MQSC command DISPLAY TOPIC to display the attributes of one or more IBM WebSphere MQ topic objects of any type.

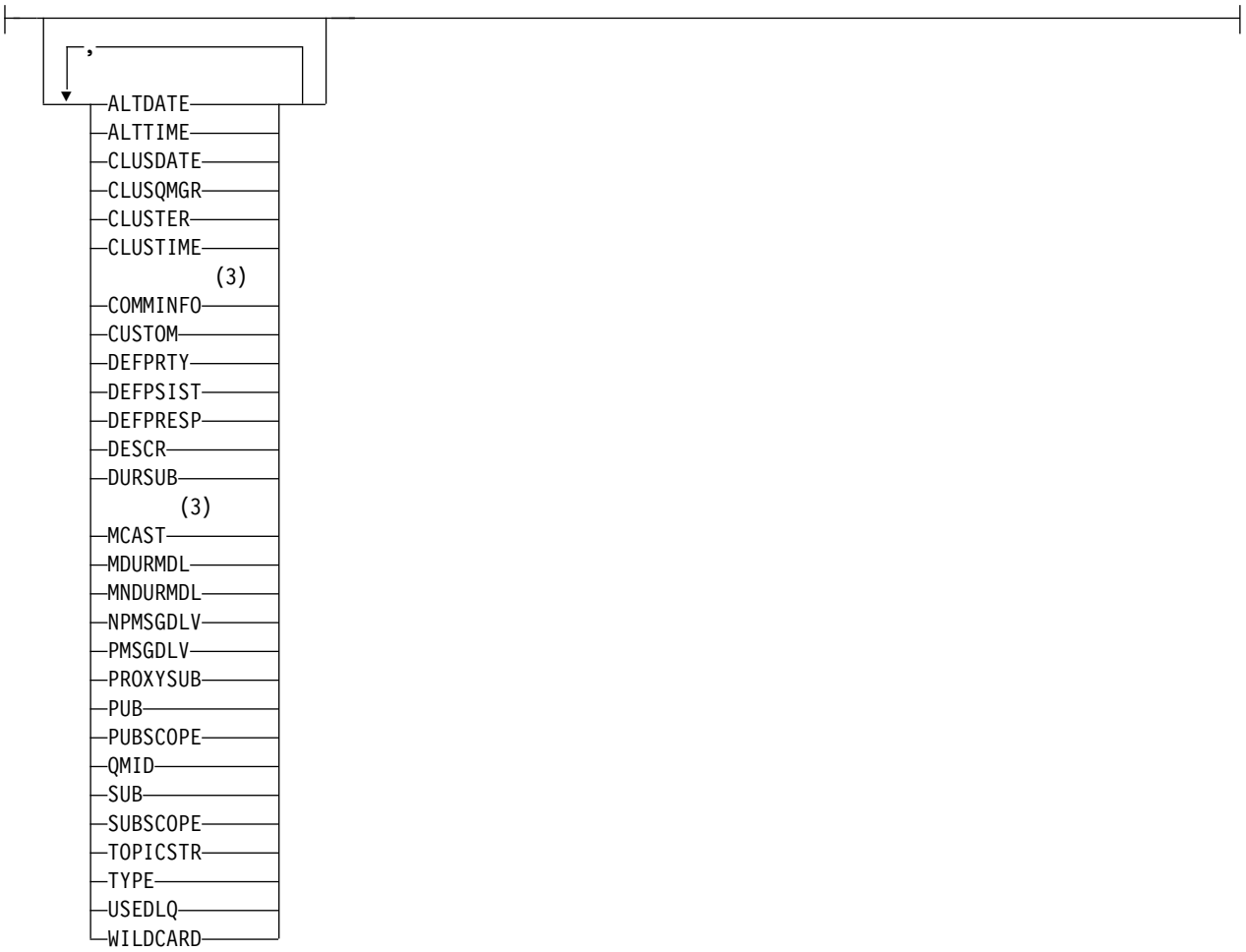
UNIX and Linux	Windows
✓	✓

Synonym: DIS TOPIC

DISPLAY TOPIC



Requested attrs:



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Usage notes for DISPLAY TOPIC

1. On z/OS, the channel initiator must be running before you can display information about cluster topics, using TYPE(CLUSTER) or the CLUSINFO parameter.
2. The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed. On z/OS, these non-printable characters are displayed as blanks. On distributed platforms using the **runmqsc** command, these non-printable characters are displayed as dots.
3. You can use the following command (or synonym) as an alternative way to display these attributes.
   - **DISPLAY TCLUSTER**

   This command produces the same output as the DISPLAY TOPIC TYPE(CLUSTER) command. If you enter the command in this way, do not use the TYPE parameter.

Parameter descriptions for DISPLAY TOPIC

You must specify the name of the topic definition you want to display. This name can be a specific topic name or a generic topic name. By using a generic topic name, you can display either:

- All topic definitions
- One or more topic definitions that match the specified name

(generic-topic-name)
The name of the administrative topic definition to be displayed (see Rules for naming IBM WebSphere MQ objects). A trailing asterisk (*) matches all administrative topic objects with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all administrative topic objects.

WHERE
Specify a filter condition to display only those administrative topic object definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Almost any parameter that can be used to display attributes for this DISPLAY command. However, you cannot use the CMDSCOPE, or QSGDISP parameters as filter keywords.

operator
This part is used to determine whether a topic object satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *filter-value*

NL Does not match a generic string that you provide as a *filter-value*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:

- An explicit value, that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter, you can use only EQ or NE.
- A generic value. This value is a character string (such as the character string you supply for the DESCR parameter) with an asterisk at the end, for example ABC*. If the operator is LK, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is NL, all items where the attribute value does not begin with the string are listed. Only a single trailing wildcard character (asterisk) is permitted.

  You cannot use a generic filter-value for parameters with numeric values or with one of a set of values.

ALL Specify this parameter to display all the attributes. If this parameter is specified, any attributes that are requested specifically have no effect; all attributes are still displayed.

This is the default if you do not specify a generic name, and do not request any specific attributes.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' ' The command is executed on the queue manager on which it was entered. This value is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this process is the same as entering the command on every queue manager in the queue-sharing group.

You cannot use CMDSCOPE as a filter keyword.

QSGDISP

Specifies the disposition of the objects for which information is to be displayed. Values are:

LIVE LIVE is the default value and displays information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

ALL Display information for objects defined with QSGDISP(QMGR) or QSGDISP(COPY).

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with QSGDISP(GROUP).

If QSGDISP(ALL) is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

In a shared queue manager environment, use

```
DISPLAY TOPIC(name) CMDSCOPE(*) QSGDISP(ALL)
```

to list ALL objects matching `name` in the queue-sharing group without duplicating those objects in the shared repository.

COPY Display information only for objects defined with QSGDISP(COPY).

GROUP
Display information only for objects defined with QSGDISP(GROUP). This is allowed only if there is a shared queue manager environment.

PRIVATE
Display information only for objects defined with QSGDISP(QMGR) or QSGDISP(COPY). QSGDISP(PRIVATE) displays the same information as QSGDISP(LIVE).

QMGR
Display information only for objects defined with QSGDISP(QMGR).

QSGDISP
QSGDISP displays one of the following values:

QMGR
The object was defined with QSGDISP(QMGR).

GROUP
The object was defined with QSGDISP(GROUP).

COPY The object was defined with QSGDISP(COPY).

You cannot use QSGDISP as a filter keyword.

CLUSINFO
Requests that, in addition to information about attributes of topics defined on this queue manager, information about these and other topics in the cluster, that match the selection criteria, is displayed. In this case, there might be multiple topics with the same topic string displayed. The cluster information is obtained from the repository on this queue manager.

On z/OS, the channel initiator must be running before you can use the CLUSINFO parameter to display information about cluster topics.

CLUSTER
Limits the information displayed to topics with the specified cluster name if entered with a value in brackets. The value can be a generic name.

If you do not enter a value to qualify this parameter, it is treated as a requested parameter, and cluster name information is returned about all the topics displayed.

On z/OS, the channel initiator must be running before you can use the CLUSINFO parameter to display information about cluster topics.

TYPE Specifies the type of topics that you want to be displayed. Values are:

ALL Display all topic types, including cluster topics if you also specify CLUSINFO.

LOCAL
Display locally defined topics.

CLUSTER
Display topics that are defined in publish/subscribe clusters. Cluster attributes include:

CLUSDATE
The date on which the definition became available to the local queue manager, in the form `yyyy-mm-dd`.

CLUSQMGR
The name of the queue manager hosting the topic.

CLUSTIME
The time at which the definition became available to the local queue manager, in the form `hh.mm.ss`.

QMID
The internally generated, unique name of the queue manager hosting the topic.

Requested parameters

Specify one or more parameters that define the data to be displayed. The parameters can be specified in any order, but do not specify the same parameter more than once.

Most of the parameters are relevant for both types of topics, but parameters that are not relevant for a particular type of topic cause no output, nor is an error raised.

The following table shows the parameters that are relevant for each type of topic. There is a brief description of each parameter after the table, but for more information, see "DEFINE TOPIC" on page 552.

Table 87. Parameters that can be returned by the DISPLAY TOPIC command

	Local topic	Cluster topic
ALTDATE	✓	✓
ALTTIME	✓	✓
CLUSDATE		✓
CLUSQMGR		✓
CLUSTER	✓	✓
CLUSTIME		✓
COMMINFO	✓	
CUSTOM	✓	✓
DEFPRTY	✓	✓
DEFPSIST	✓	✓
DEFPRESP	✓	✓
DESCR	✓	✓
DURSUB	✓	✓
MCAST	✓	
MDURMDL	✓	✓
MNDURMDL	✓	✓
NPMSGDLV	✓	✓
PMSGDLV	✓	✓

Table 87. Parameters that can be returned by the DISPLAY TOPIC command (continued)

	Local topic	Cluster topic
PROXYSUB	✓	✓
PUB	✓	✓
PUBSCOPE	✓	✓
QMID		✓
SUB	✓	✓
SUBSCOPE	✓	✓
TOPICSTR	✓	✓
TYPE	✓	✓
USEDLQ	✓	
WILDCARD	✓	✓

ALTDATE
The date on which the definition or information was last altered, in the form `yyyy-mm-dd`.

ALTTIME
The time at which the definition or information was last altered, in the form `hh.mm.ss`.

CLUSDATE
The date on which the information became available to the local queue manager, in the form `yyyy-mm-dd`.

CLUSQMGR
The name of the queue manager that hosts the topic.

CLUSTER
The name of the cluster that the topic is in.

CLUSTIME
The time at which the information became available to the local queue manager, in the form `hh.mm.ss`.

COMMINFO
The communication information object name.

CUSTOM
This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value in the form `NAME(VALUE)`.

DEFPRTY
Default priority of the messages published to this topic.

DEFPSIST
Default persistence of messages published to this topic.

DEFPRESP
Default put response for this topic. This attribute defines the behavior that must be used by applications when the put response type in the MQPMO options has been set to MQPMO_RESPONSE_AS_TOPIC_DEF.

DESCR
Description of this administrative topic object.

DURSUB
Determines whether the topic permits durable subscriptions to be made.

MCAST
Specifies whether the topic is enabled for multicast.

MDURMDL
The name of the model queue for durable managed subscriptions.

MNDURMDL
The name of the model queue for non-durable managed subscriptions.

NPMSGDLV
The delivery mechanism for non-persistent messages.

PMSGDLV
The delivery mechanism for persistent messages.

PROXYSUB
Determines whether a proxy subscription is forced for this subscription, even if no local subscriptions exist.

PUB Determines whether the topic is enabled for publication.

PUBSCOPE
Determines whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster.

QMID
The internally generated unique name of the queue manager that hosts the topic.

SUB Determines whether the topic is enabled for subscription.

SUBSCOPE
Determines whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster.

TOPICSTR
The topic string.

TYPE Specifies whether this object is a local topic or cluster topic.

USEDLQ
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue.

WILDCARD
The behavior of wildcard subscriptions with respect to this topic.

For more details of these parameters, see “DEFINE TOPIC” on page 552.

Related reference:
"DISPLAY TPSTATUS"
Use the MQSC command DISPLAY TPSTATUS to display the status of one or more topics in a topic tree.

DISPLAY TPSTATUS:

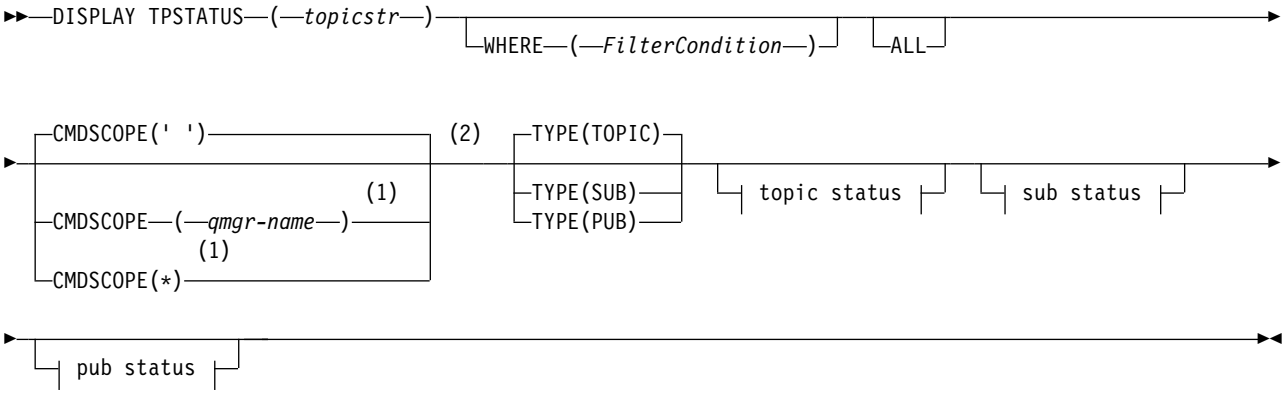
Use the MQSC command DISPLAY TPSTATUS to display the status of one or more topics in a topic tree.

UNIX and Linux	Windows
✓	✓

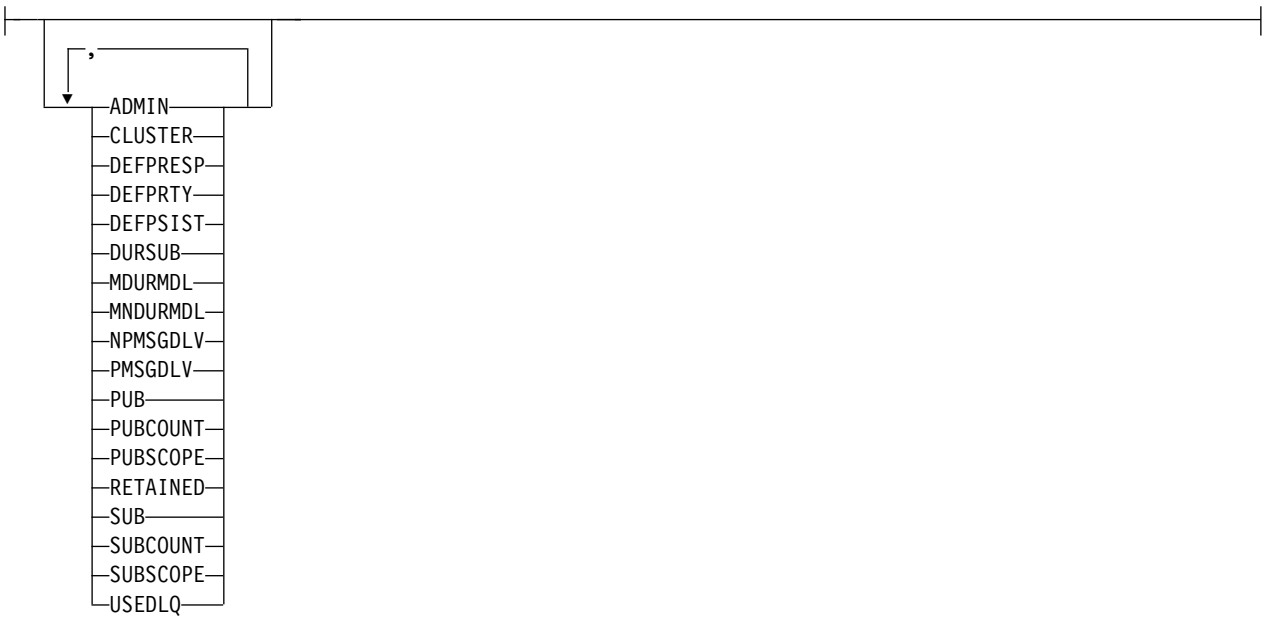
- Syntax diagram
- "Usage notes for DISPLAY TPSTATUS" on page 739
- "Parameter descriptions for DISPLAY TPSTATUS" on page 739
- "Topic status parameters" on page 741
- "Sub status parameters" on page 742
- "Pub status parameters" on page 743

Synonym: DIS TPS

DISPLAY TPSTATUS



Topic status:



Sub status:

Pub status:

Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

3 Not valid on z/OS.

Usage notes for DISPLAY TPSTATUS

1. The TOPICSTR parameter might contain characters that cannot be translated into printable characters when the command output is displayed. On z/OS, these non-printable characters are displayed as blanks. On distributed platforms using the **runmqsc** command, these non-printable characters are displayed as dots.
2. The topic-string input parameter on this command must match the topic you want to act on. Keep the character strings in your topic strings as characters that can be used from the location issuing the command. If you issue commands using MQSC, you have fewer characters available to you than if you are using an application that submits PCF messages, such as the WebSphere MQ Explorer.

Parameter descriptions for DISPLAY TPSTATUS

The DISPLAY TPSTATUS command requires a topic string value to determine which topic nodes the command returns.

(topicstr)
The value of the topic string for which you want to display status information. You cannot specify the name of a WebSphere MQ topic object.

The topic string can have one of the following values:

- A specific topic string value. For example, `DIS TPS('Sports/Football')` returns just the 'Sports/Football' node.
- A topic string containing a "+" wildcard character. For example, `DIS TPS('Sports/Football/+')` returns all direct child nodes of the 'Sports/Football' node.
- A topic string containing a "#" wildcard character. For example, `DIS TPS('Sports/Football/#')` returns the 'Sports/Football' node and all its descendant nodes.
- A topic string containing more than one wildcard. For example, `DIS TPS('Sports/+/Teams/#')` returns any direct child node of 'Sports' that also has a 'teams' child, with all descendants of the latter nodes.

The **DISPLAY TPSTATUS** command does not support the '*' wildcard. For more information about using wildcards, see the related topic.

- To return a list of all root-level topics, use `DIS TPS('+')`
- To return a list of all topics in the topic tree, use `DIS TPS('#')`, but note that this command might return a large amount of data.
- To filter the list of topics returned, use the **WHERE** parameter. For example, `DIS TPS('Sports/Football/+') WHERE(TOPICSTR LK 'Sports/Football/L*')` returns all direct child nodes of the 'Sports/Football' node, that begin with the letter "L".

WHERE
Specifies a filter condition to display only those administrative topic definitions that satisfy the selection criterion of the filter condition. The filter condition is in three parts: *filter-keyword*, *operator*, and *filter-value*:

filter-keyword
Except for the CMDSCOPE parameter, any parameter that you can use with this DISPLAY command.

operator
Determines whether a topic string satisfies the filter value on the given filter keyword. The operators are:

LT Less than

GT Greater than

EQ Equal to

NE Not equal to

LE Less than or equal to

GE Greater than or equal to

LK Matches a generic string that you provide as a *topicstr*

NL Does not match a generic string that you provide as a *topicstr*

filter-value
The value that the attribute value must be tested against using the operator. Depending on the filter-keyword, this value can be:

- An explicit value that is a valid value for the attribute being tested.

  You can use operators LT, GT, EQ, NE, LE, or GE only. However, if the attribute value is one from a possible set of values on a parameter, you can use only EQ or NE.
- A generic value. This value is a character string with an asterisk at the end, for example ABC*. If the operator is LK, the command lists all topic nodes that begin with the string (ABC in the example). If the operator is NL, the command lists all topic nodes that do not begin with the string.

  You cannot use a generic *filter-value* for parameters with numeric values or with one of a set of values.

ALL Use this parameter to display all attributes.

If this parameter is specified, any attributes that you request specifically have no effect; the command displays all attributes.

This parameter is the default parameter if you do not specify a generic name, and do not request any specific attributes.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command runs when the queue manager is a member of a queue-sharing group.

' ' The command runs on the queue manager on which it was entered. This value is the default value.

qmgr-name
The command runs on the named queue manager, if the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which you enter the command, but only if you are using a queue-sharing group environment and the command server is enabled.

***** The command runs on the local queue manager and on every active queue manager in the queue-sharing group. The effect of this option is equivalent to entering the command on every queue manager in the queue-sharing group.

TYPE

TOPIC
The command displays status information relating to each topic node, which is the default if you do not provide a **TYPE** parameter.

PUB The command displays status information relating to applications that have topic nodes open for publish.

SUB The command displays status information relating to applications that subscribe to the topic node or nodes. The subscribers that the command returns are not necessarily the subscribers that would receive a message published to this topic node. The value of SelectionString or SubLevel determines which subscribers receive such messages.

Topic status parameters

Topic status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

ADMIN
If the topic node is an admin-node, the command displays the associated topic object name containing the node configuration. If the field is not an admin-node the command displays a blank.

CLUSTER
The name of the cluster to which this topic belongs.

' ' This topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers.

DEFPRESP
Displays the resolved default put response of messages published to the topic, if it has no *ASPARENT* response value. The value can be *SYNC* or *ASYNC*

DEFPRTY
Displays the resolved default priority of messages published to the topic, if it has no *ASPARENT* response value.

DEFPSIST
Displays the resolved default persistence for this topic string, if it has no *ASPARENT* response value. The value can be *YES* or *NO*.

DURSUB
Displays the resolved value that shows whether applications can make durable subscriptions, if there is no *ASPARENT* response value. The value can be *YES* or *NO*.

MDURMDL
Displays the resolved value of the name of the model queue to be used for durable subscriptions. The name cannot be blank, because that is the equivalent of *ASPARENT* for this parameter.

MNDURMDL
Displays the resolved value of the name of the model queue used for non-durable subscriptions. The name cannot be blank, because that is the equivalent of *ASPARENT* for this parameter.

NPMSGDLV
Displays the resolved value for the delivery mechanism for non-persistent messages published to this topic. The value can be *ALL*, *ALLDUR*, or *ALLAVAIL*, but not ASPARENT.

PMSGDLV
Displays the resolved value for the delivery mechanism for persistent messages published to this topic. The value can be *ALL*, *ALLDUR*, or *ALLAVAIL*, but not ASPARENT.

PUB Displays the resolved value that shows whether publications are allowed for this topic, if there is no *ASPARENT* response value. The values can be *ENABLED* or *DISABLED*.

PUBCOUNT
Displays the number of handles that are open for publish on this topic node.

PUBSCOPE
Determines whether this queue manager propagates publications, for this topic node, to queue managers as part of a hierarchy or as part of a publish/subscribe operation. The value can be *QMGR* or *ALL*.

RETAINED
Displays whether there is a retained publication associated with this topic. The value can be *YES* or *NO*.

SUB Displays the resolved value that shows whether subscriptions are allowed for this topic, if there is no ASPARENT response value. The values can be *ENABLED* or *DISABLED*.

SUBCOUNT
Displays the number of subscribers to this topic node, including durable subscribers that are not currently connected.

SUBSCOPE
Determines whether this queue manager propagates subscriptions, for this topic node, to other queue managers as part of a hierarchy or cluster, or whether it restricts the subscriptions to only the local queue manager. The value can be *QMGR* or *ALL*.

USEDLQ
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue. The value can be *YES* or *NO*.

Sub status parameters

Sub status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

ACTCONN
Detects local publications, returning the currently active ConnectionId (CONNID) that opened this subscription.

DURABLE
Indicates whether a durable subscription is not deleted when the creating application closes its subscription handle, and persists over queue manager restart. The value can be *YES* or *NO*.

LMSGDATE
The date on which an MQPUT call last sent a message to this subscription. The MQPUT call updates the date field only when the call successfully puts a message to the destination specified by this subscription. An MQSUBRQ call causes an update to this value.

LMSGTIME
The time at which an MQPUT call last sent a message to this subscription. The MQPUT call updates the time field only when the call successfully puts a message to the destination specified by this subscription. An MQSUBRQ call causes an update to this value.

MCASTREL
Indicator of the reliability of the multicast messages.

The values are expressed as a percentage. A value of 100 indicates that all messages are being delivered without problems. A value less than 100 indicates that some of the messages are experiencing network issues. To determine the nature of these issues the user can switch on event message generation, use the **COMMEV** parameter of the COMMINFO objects, and examine the generated event messages.

The following two values are returned:

- The first value is based on recent activity over a short period.
- The second value is based on activity over a longer period.

If no measurement is available the values are shown as blanks.

NUMMSGS
Number of messages put to the destination specified by this subscription. An MQSUBRQ call causes an update to this value.

RESMDATE
Date of the most recent MQSUB call that connected to this subscription.

RESMTIME
Time of the most recent MQSUB call that connected to this subscription.

SUBID
An all time unique identifier for this subscription, assigned by the queue manager. The format of **SUBID** matches that of a CorrelId. For durable subscriptions, the command returns the **SUBID** even if the subscriber is not currently connected to the queue manager.

SUBTYPE
The type of subscription, indicating how it was created. The value can be *ADMIN*, *API*, or *PROXY*.

SUBUSER
The user ID that owns this subscription, which can be either the user ID associated with the creator of the subscription or, if subscription takeover is permitted, the user ID that last took over the subscription.

Pub status parameters

Pub status parameters define the data that the command displays. You can specify these parameters in any order but must not specify the same parameter more than once.

ACTCONN
The currently active ConnectionId (CONNID) associated with the handle that has this topic node open for publish.

LPUBDATE
The date on which this publisher last sent a message.

LPUBTIME
The time at which this publisher last sent a message.

MCASTREL
Indicator of the reliability of the multicast messages.

The values are expressed as a percentage. A value of 100 indicates that all messages are being delivered without problems. A value less than 100 indicates that some of the messages are experiencing network issues. To determine the nature of these issues the user can switch on event message generation, using the **COMMEV** parameter of the COMMINFO objects, and examine the generated event messages.

The following two values are returned:

- The first value is based on recent activity over a short period.
- The second value is based on activity over a longer period.

If no measurement is available the values are shown as blanks.

NUMPUBS
Number of publishes by this publisher. This value records the actual number of publishes, not the total number of messages published to all subscribers.

PING CHANNEL:

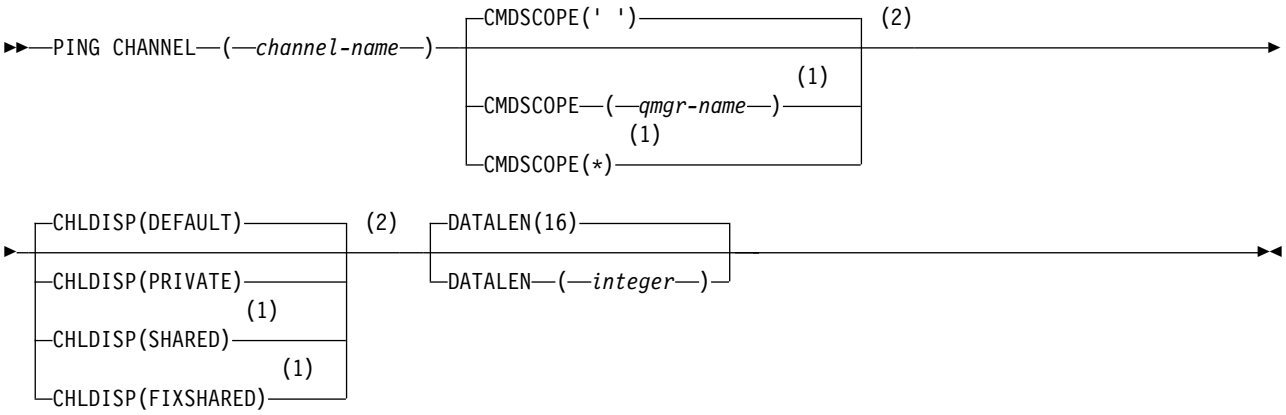
Use the MQSC command PING CHANNEL to test a channel by sending data as a special message to the remote queue manager, and checking that the data is returned. The data is generated by the local queue manager.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for PING CHANNEL"

Synonym: PING CHL

PING CHANNEL



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

1. On z/OS, the command server and the channel initiator must be running.
2. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager's repository.
3. This command can be used only for sender (SDR), server (SVR), and cluster-sender (CLUSSDR) channels (including those that have been defined automatically). It is not valid if the channel is running; however, it is valid if the channel is stopped or in retry mode.

Parameter descriptions for PING CHANNEL

(channel-name)
The name of the channel to be tested. This is required.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

Note: The '*' option is not permitted if CHLDISP is FIXSHARED.

CHLDISP
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED
- FIXSHARED

If this parameter is omitted, then the DEFAULT value applies. This is the value of the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is **not** related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table.

Table 88. CHLDISP and CMDSCOPE for PING CHANNEL

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	Ping private channel on the local queue manager	Ping private channel on the named queue manager	Ping private channel on all active queue managers
SHARED	Ping a shared channel on the most suitable queue manager in the group This might automatically generate a command using CMDSCOPE and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted
FIXSHARED	Ping a shared channel on the local queue manager	Ping a shared channel on the named queue manager	Not permitted

DATALEN(*integer***)**
The length of the data, in the range 16 through 32 768. This is optional.

PING QMGR:

Use the MQSC command PING QMGR to test whether the queue manager is responsive to commands.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes"

Synonym: PING QMGR

PING QMGR

Usage notes

If commands are issued to the queue manager by sending messages to the command server queue, this command causes a special message to be sent to it, consisting of a command header only, and checking that a positive reply is returned.

PURGE CHANNEL:

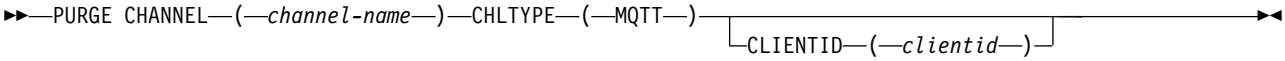
Use the MQSC command PURGE CHANNEL to stop and purge a telemetry channel. Purging a telemetry channel disconnects all the MQTT clients connected to it, cleans up the state of the MQTT clients, and stops the telemetry channel. Cleaning the state of a client deletes all the pending publications and removes all the subscriptions from the client.

IBM i	UNIX and Linux	Windows	z/OS
	✓	✓	

- Syntax diagram
- "Parameter descriptions for PURGE CHANNEL"

Synonym: None

PURGE CHANNEL



Parameter descriptions for PURGE CHANNEL

(channel name)
The name of the telemetry channel to be stopped and purged. This parameter is required.

CHLTYPE*(MQTT)*
Channel type. This parameter is required. It must follow immediately after the (channel-name) parameter on all platforms except z/OS, and the value must currently be `MQTT`.

CLIENTID*(string)*
Client identifier. The client identifier is a 23 byte string that identifies a IBM WebSphere MQ Telemetry Transport client. When the PURGE CHANNEL command specifies a CLIENTID, only the connection for the specified client identifier is purged. If the CLIENTID is not specified, all the connections on the channel are purged.

REFRESH CLUSTER:

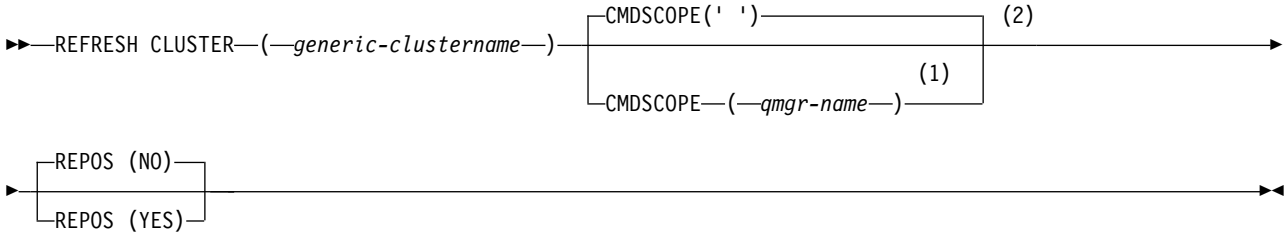
Use the MQSC command **REFRESH CLUSTER** to discard all locally held cluster information and force it to be rebuilt. The command also processes any autodefined channels that are in doubt. After the command completes processing, you can perform a "cold-start" on the cluster.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage Notes for REFRESH CLUSTER" on page 748
- "Parameter descriptions for REFRESH CLUSTER" on page 749

Synonym: `REF CLUSTER`

REFRESH CLUSTER



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage Notes for REFRESH CLUSTER

1. Issuing **REFRESH CLUSTER** is disruptive to the cluster. It might make cluster objects invisible for a short time until the **REFRESH CLUSTER** processing completes. Specifically, if an application is using a queue that it opened with MQOO_BIND_NOT_FIXED, it might receive the return code MQRC_NO_DESTINATIONS_AVAILABLE. If an application is publishing or subscribing on a cluster topic, that topic might become temporarily unavailable. The unavailability results in a pause in the publication stream until the **REFRESH CLUSTER** command completes. If the command is issued on a full repository queue manager, **REFRESH CLUSTER** might make a large volume of messages flow.
2. For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.
3. Quiesce all publish/subscribe applications before issuing the **REFRESH CLUSTER** command, because issuing this command in a publish/subscribe cluster disrupts delivery of publications to and from other queue managers in the cluster, and might result in proxy subscriptions from other queue managers being cancelled. If this happens, use REFRESH QMGR TYPE(PROXYSUB) to resynchronize after the cluster has refreshed, and keep all publish/subscribe applications quiesced until after the proxy subscriptions have been resynchronized. See REFRESH CLUSTER considerations for publish/subscribe clusters.
4. When the command returns control to the user, it does not signify the command has completed. Activity on SYSTEM.CLUSTER.COMMAND.QUEUE indicates the command is still processing.
5. If cluster-sender channels are running at the time **REFRESH CLUSTER** is issued, the refresh might not complete until the channels stop and restart. To hasten completion, stop all cluster-sender channels for the cluster before you run the **REFRESH CLUSTER** command. During the processing of the **REFRESH CLUSTER** command, if the channel is not in doubt, the channel state might be recreated.
6. If you select **REPOS(YES)**, check that all cluster-sender channels in the relevant cluster are inactive or stopped before you issue the **REFRESH CLUSTER** command.

   If cluster-sender channels are running at the time you run the **REFRESH CLUSTER REPOS(YES)**command, those cluster-sender channels are ended during the operation and left in an INACTIVE state after the operation completes. Alternatively, you can force the channels to stop using the STOP CHANNEL command with MODE(FORCE).

   Stopping the channels ensures that the refresh can remove the channel state, and that the channel runs with the refreshed version after the refresh completes. If the state of a channel cannot be deleted, its state is not renewed after the refresh. If a channel was stopped, it does not automatically restart. The channel state cannot be deleted if the channel is in doubt, or because it is also running as part of another cluster.

   If you choose the option **REPOS(YES)** on full repository queue manager, you must alter it to be a partial repository. If it is the sole working repository in the cluster, the result is that there is no full

repository left in the cluster. After the queue manager is refreshed, and restored to its status of a full repository, you must refresh the other partial repositories to restore a working cluster.

If it is not the sole remaining repository, you do not need to refresh the partial repositories manually. Another working full repository in the cluster informs the other members of the cluster that the full repository running the **REFRESH CLUSTER** command resumed its role as a full repository.

7. It is not normally necessary to issue a **REFRESH CLUSTER** command except in one of the following circumstances:
   - Messages were removed from either the SYSTEM.CLUSTER.COMMAND.QUEUE, or from another a cluster transmission queue, where the destination queue is SYSTEM.CLUSTER.COMMAND.QUEUE on the queue manager in question.
   - Issuing a **REFRESH CLUSTER** command is recommended by IBM Service.
   - The CLUSRCVR channels were removed from a cluster, or their CONNAMEs were altered on two or more full repository queue managers while they could not communicate.
   - The same name was used for a CLUSRCVR channel on more than one queue manager in a cluster. As a result, messages destined for one of the queue managers were delivered to another. In this case, remove the duplicates, and run a **REFRESH CLUSTER** command on the single remaining queue manager with a CLUSRCVR definition.
   - RESET CLUSTER ACTION(FORCEREMOVE) was issued in error.
   - The queue manager was restarted from an earlier point in time than it finished last time it was used; for example, by restoring backed up data.
8. Issuing **REFRESH CLUSTER** does not correct mistakes in cluster definitions, nor is it necessary to issue the command after such mistakes are corrected.
9. During **REFRESH CLUSTER** processing, the queue manager generates the message AMQ9875 followed by the message AMQ9442 or AMQ9404. The queue manager might also generate the message AMQ9420. If the cluster functionality is not affected, the message AMQ9420 can be ignored.
10. On UNIX systems, the command is valid only on AIX, HP-UX, Linux, and Solaris.
11. On z/OS, the command fails if the channel initiator is not started.
12. On z/OS, any errors are reported to the console on the system where the channel initiator is running. They are not reported to the system that issued the command.

Parameter descriptions for REFRESH CLUSTER

(*generic-clustername***)**
The name of the cluster to be refreshed. Alternatively *generic-clustername* can be specified as "*". If "*" is specified, the queue manager is refreshed in all the clusters that it is a member of. If used with **REPOS(YES)**, this forces the queue manager to restart its search for full repositories from the information in the local CLUSSDR definitions. It restarts its search, even if the CLUSSDR definitions connect the queue manager to several clusters.

The *generic-clustername* parameter is required.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

'' The command is executed on the queue manager on which it was entered. '' is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered. If you do so, you must be using a queue-sharing group environment and the command server must be enabled.

REPOS

Specifies whether objects representing full repository cluster queue managers are also refreshed.

NO The queue manager retains knowledge of all cluster queue manager and cluster queues marked as locally defined. It also retains knowledge of all cluster queue managers that are marked as full repositories. In addition, if the queue manager is a full repository for the cluster, it retains knowledge of the other cluster queue managers in the cluster. Everything else is removed from the local copy of the repository and rebuilt from the other full repositories in the cluster. Cluster channels are not stopped if **REPOS(NO)** is used. A full repository uses its CLUSSDR channels to inform the rest of the cluster that it completed its refresh.

NO is the default.

YES Specifies that in addition to the **REPOS(NO)** behavior, objects representing full repository cluster queue managers are also refreshed. The **REPOS(YES)** option must not be used if the queue manager is itself a full repository. If it is a full repository, you must first alter it so that it is not a full repository for the cluster in question. The full repository location is recovered from the manually defined CLUSSDR definitions. After the refresh with **REPOS(YES)** is issued, the queue manager can be altered so that it is once again a full repository, if required.

On z/OS, N and Y are accepted synonyms of NO and YES.

Related information:

Clustering: Using REFRESH CLUSTER best practices

REFRESH QMGR:

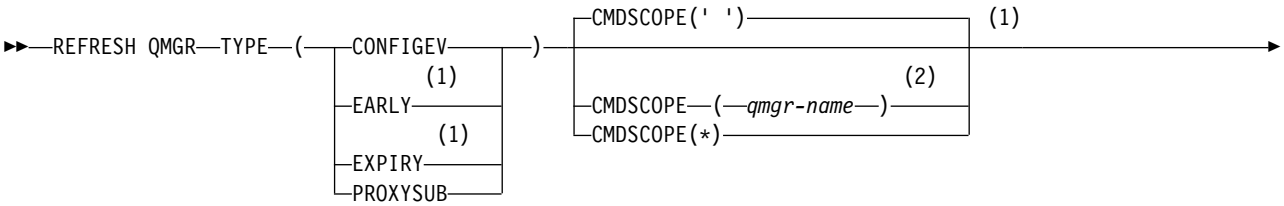
Use the MQSC command REFRESH QMGR to perform special operations on queue managers.

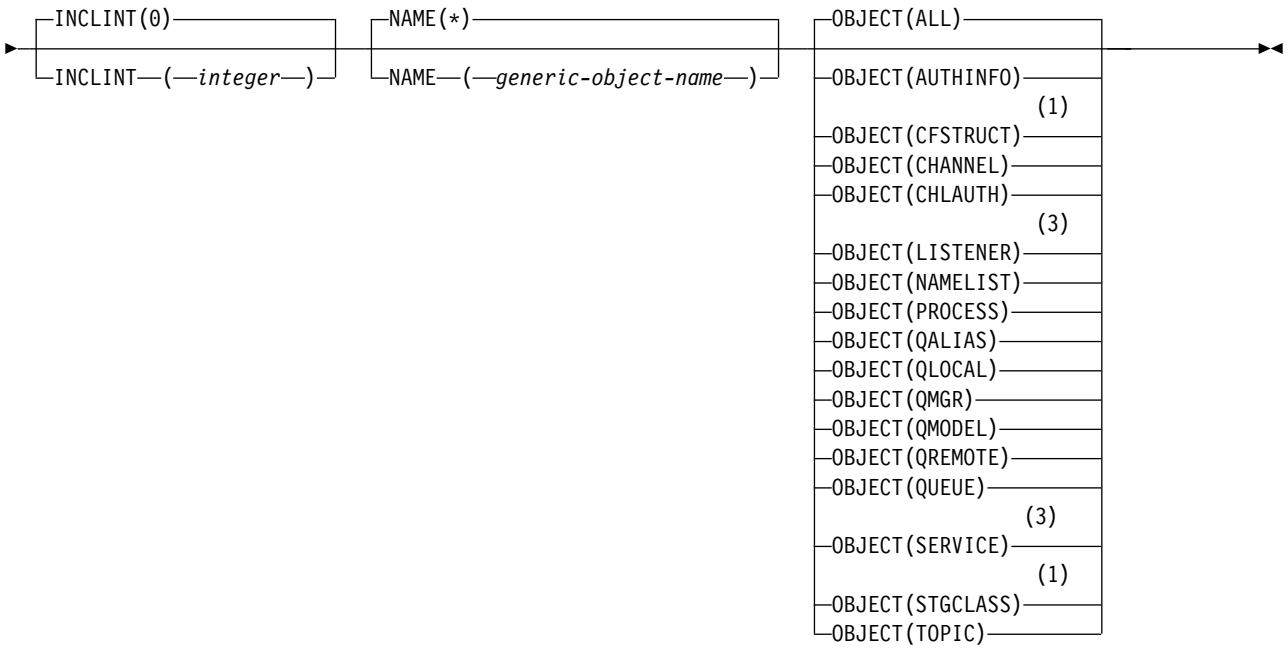
UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage Notes for REFRESH QMGR" on page 751
- "Parameter descriptions for REFRESH QMGR" on page 751

Synonym: None

REFRESH QMGR





Notes:

1 Valid only on z/OS.

2 Valid only when the queue manager is a member of a queue-sharing group.

3 Not valid on z/OS.

Usage Notes® for REFRESH QMGR

1. Issue this command with TYPE(CONFIGEV) after setting the CONFIGEV queue manager attribute to ENABLED, to bring the queue manager configuration up to date. To ensure that complete configuration information is generated, include all objects; if you have many objects, it might be preferable to use several commands, each with a different selection of objects, but such that all are included.
2. You can also use the command with TYPE(CONFIGEV) to recover from problems such as errors on the event queue. In such cases, use appropriate selection criteria, to avoid excessive processing time and event messages generation.
3. Issue the command with TYPE(EXPIRY) at any time when you believe that a queue could contain numbers of expired messages.
4. You are unlikely to use REFRESH QMGR TYPE(PROXYSUB) other than in exceptional circumstances. Typically, a queue manager revalidates proxy subscriptions with affected directly-connected queue managers as follows:
   - When forming a hierarchical connection
   - When modifying the PUBSCOPE or SUBSCOPE or CLUSTER attributes on a topic object
   - When restarting the queue manager

Parameter descriptions for REFRESH QMGR

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

'' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

This parameter is not valid with TYPE(EARLY).

INCLINT*(integer)*
Specifies a value in minutes defining a period immediately before the current time, and requests that only objects that have been created or changed within that period (as defined by the ALTDATE and ALTTIME attributes) are included. The value must be in the range zero through 999 999. A value of zero means there is no time limit (this is the default).

This parameter is valid only with TYPE(CONFIGEV).

NAME*(generic-object-name)*
Requests that only objects with names that match the one specified are included. A trailing asterisk (*) matches all object names with the specified stem followed by zero or more characters. An asterisk (*) on its own specifies all objects (this is the default). NAME is ignored if OBJECT(QMGR) is specified.

This parameter is not valid with TYPE(EARLY).

OBJECT*(objtype)*
Requests that only objects of the specified type are included. (Synonyms for object types, such as QL, can also be specified.) The default is ALL, to include objects of every type.

This parameter is valid only with TYPE(CONFIGEV).

TYPE This is required. Values are:

CONFIGEV
Requests that the queue manager generates a configuration event message for every object that matches the selection criteria specified by the OBJECT, NAME and INCLINT parameters. Matching objects defined with QSGDISP(QMGR) or QSGDISP(COPY) are always included. Matching objects defined with QSGDISP(GROUP) or QSGDISP(SHARED) are included only if the command is being executed on the queue manager where it is entered.

EARLY
Requests that the subsystem function routines (generally known as early code) for the queue manager replace themselves with the corresponding routines in the linkpack area (LPA).

You need to use this command only after you install new subsystem function routines (provided as corrective maintenance or with a new version or release of WebSphere MQ). This command instructs the queue manager to use the new routines.

EXPIRY
Requests that the queue manager performs a scan to discard expired messages for every queue that matches the selection criteria specified by the NAME parameter. (The scan is performed regardless of the setting of the EXPRYINT queue manager attribute.)

PROXYSUB

Requests that the queue manager resynchronizes the proxy subscriptions that are held with, and on behalf of, queue managers that are connected in a hierarchy or publish/subscribe cluster.

You must resynchronize the proxy subscriptions only in exceptional circumstances, for example, when the queue manager is receiving subscriptions that it must not be sent, or not receiving subscriptions that it must receive. The following list describes some of the exceptional reasons for resynchronizing proxy subscriptions:

- Disaster recovery.
- Problems that are identified in a queue manager error log where messages inform of the issuing of the REFRESH QMGR TYPE(REPOS) command.
- Operator errors, for example, issuing a DELETE SUB command on a proxy subscription.

Missing proxy subscriptions can be caused if the closest matching topic definition is specified with **Subscription scope** set to `Queue Manager` or it has an empty or incorrect cluster name. Note that **Publication scope** does not prevent the sending of proxy subscriptions, but does prevent publications from being delivered to them.

Extraneous proxy subscriptions can be caused if the closest matching topic definition is specified with **Proxy subscription behavior** set to `Force`.

Missing or extraneous proxy subscriptions that are due to configuration errors are not changed by issuing a resynchronization. A resynchronization does resolve missing or extraneous publications as a result of the exceptional reasons listed.

Note: If TYPE(EARLY) is specified, no other keywords are allowed and the command can be issued only from the z/OS console and only if the queue manager is not active.

REFRESH SECURITY:

Use the MQSC command REFRESH SECURITY to perform a security refresh.

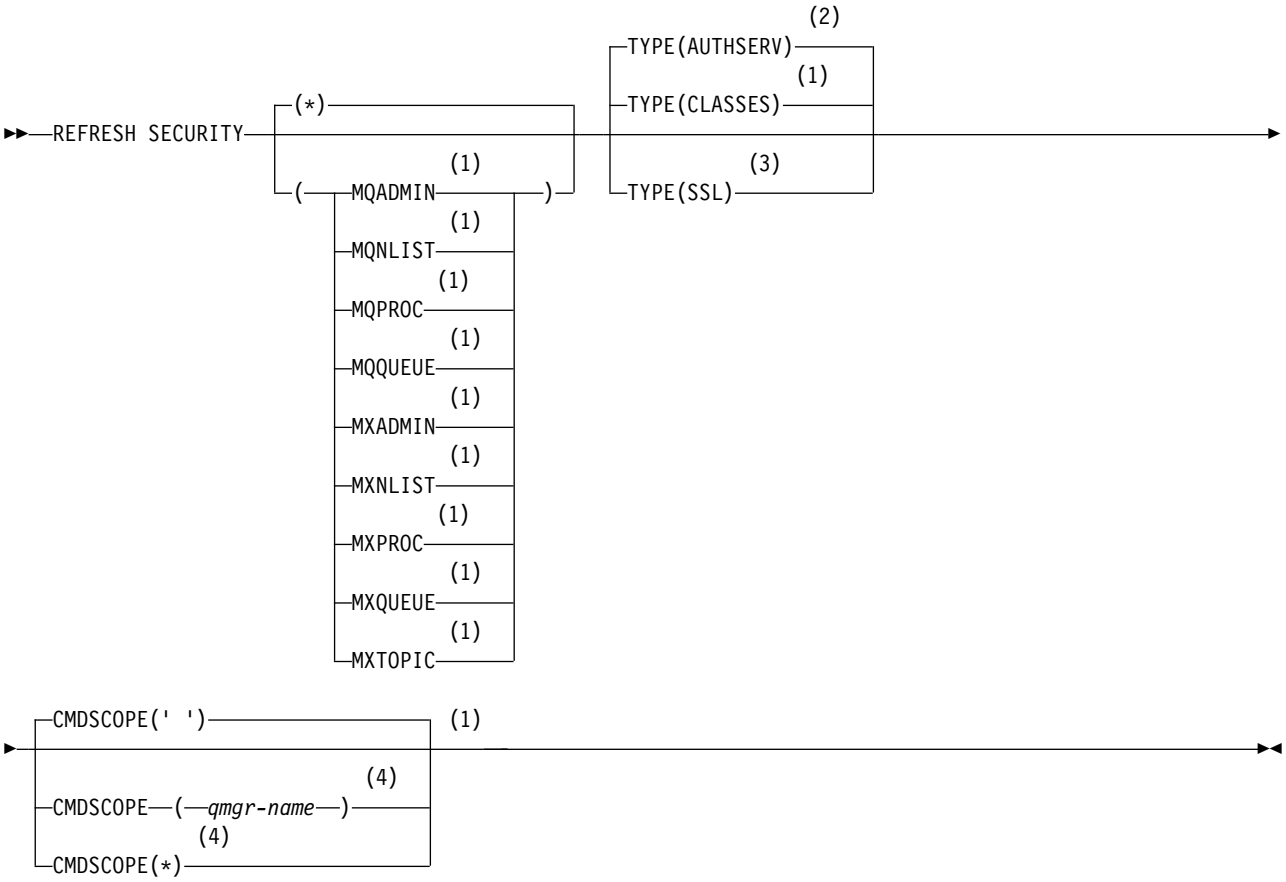
UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes for REFRESH SECURITY" on page 754
- "Parameter descriptions for REFRESH SECURITY" on page 755

Synonym: REF SEC

REBUILD SECURITY is another synonym for REFRESH SECURITY.

REFRESH SECURITY



Notes:

1 Valid only on z/OS.

2 Not valid on z/OS.

3 On WebSphere MQ for z/OS, you cannot issue this from CSQINP2.

4 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

Usage notes for REFRESH SECURITY

When you issue the REFRESH SECURITY TYPE(SSL) MQSC command, all running SSL channels are stopped and restarted. Sometimes SSL channels can take a long time to shut down and this means that the refresh operation takes some time to complete. There is a time limit of 10 minutes for an SSL refresh to complete (or 1 minute on z/OS), so it can potentially take 10 minutes for the command to finish. This can give the appearance that the refresh operation has "frozen". The refresh operation will fail with an MQSC error message of AMQ9710 or PCF error MQRCCF_COMMAND_FAILED if the timeout is exceeded before all channels have stopped. This is likely to happen if the following conditions are true:

- The queue manager has many SSL channels running simultaneously when the refresh command is invoked
- The channels are handling large numbers of messages

If a refresh fails under these conditions, retry the command later when the queue manager is less busy. In the case where many channels are running, you can choose to stop some of the channels manually before invoking the REFRESH command.

When using TYPE(SSL):

1. On z/OS, the command server and channel initiator must be running.
2. On z/OS, WebSphere MQ determines whether a refresh is needed due to one, or more, of the following reasons:
   - The contents of the key repository have changed
   - The location of the LDAP server to be used for Certification Revocation Lists has changed
   - The location of the key repository has changed

   If no refresh is needed, the command completes successfully and the channels are unaffected.
3. On platforms other than z/OS, the command updates all SSL channels regardless of whether a security refresh is needed.
4. If a refresh is to be performed, the command updates all SSL channels currently running, as follows:
   - Sender, server and cluster-sender channels using SSL are allowed to complete the current batch. In general they then run the SSL handshake again with the refreshed view of the SSL key repository. However, you must manually restart a requester-server channel on which the server definition has no CONNAME parameter.
   - All other channel types using SSL are stopped with a STOP CHANNEL MODE(FORCE) STATUS(INACTIVE) command. If the partner end of the stopped message channel has retry values defined, the channel retries and the new SSL handshake uses the refreshed view of the contents of the SSL key repository, the location of the LDAP server to be used for Certification Revocation Lists, and the location of the key repository. In the case of a server-connection channel, the client application loses its connection to the queue manager and has to reconnect in order to continue.

When using TYPE(CLASSES):

- Classes MQADMIN, MQNLIST, MQPROC, and MQQUEUE can only hold profiles defined in uppercase.
- Classes MXADMIN, MXNLIST, MXPROC, and MQXUEUE can hold profiles defined in mixed case.
- Class MXTOPIC can be refreshed whether using uppercase or mixed case classes. Although it is a mixed case class, it is the only mixed case class that can be active with either group of classes.

Notes:

1. Performing a `REFRESH SECURITY(*) TYPE(CLASSES)` operation is the only way to change the classes being used by your system from uppercase-only support to mixed case support.

   Do this by checking the queue manager attribute SCYCASE to see if it is set to UPPER or MIXED
2. It is your responsibility to ensure that you have copied, or defined, all the profiles you need in the appropriate classes before you carry out a `REFRESH SECURITY(*) TYPE(CLASSES)` operation.
3. A refresh of an individual class is allowed only if the classes currently being used are of the same type. For example, if MQPROC is in use, you can issue a refresh for MQPROC but not MXPROC.

Parameter descriptions for REFRESH SECURITY

The command qualifier allows you to indicate more precise behavior for a specific TYPE value. Select from:

* A full refresh of the type specified is performed. This is the default value.

MQADMIN
Valid only if TYPE is CLASSES. Specifies that Administration type resources are to be refreshed. Valid on z/OS only.

Note: If, when refreshing this class, it is determined that a security switch relating to one of the other classes has been changed, a refresh for that class also takes place.

MQNLIST
Valid only if TYPE is CLASSES. Specifies that Namelist resources are to be refreshed. Valid on z/OS only.

MQPROC
Valid only if TYPE is CLASSES. Specifies that Process resources are to be refreshed. Valid on z/OS only.

MQQUEUE
Valid only if TYPE is CLASSES. Specifies that Queue resources are to be refreshed. Valid on z/OS only.

MXADMIN
Valid only if TYPE is CLASSES. Specifies that administration type resources are to be refreshed. Valid on z/OS only.

Note: If, when refreshing this class, it is determined that a security switch relating to one of the other classes has been changed, a refresh for that class also takes place.

MXNLIST
Valid only if TYPE is CLASSES. Specifies that namelist resources are to be refreshed. Valid on z/OS only.

MXPROC
Valid only if TYPE is CLASSES. Specifies that process resources are to be refreshed. Valid on z/OS only.

MXQUEUE
Valid only if TYPE is CLASSES. Specifies that queue resources are to be refreshed. Valid on z/OS only.

MXTOPIC
Valid only if TYPE is CLASSES. Specifies that topic resources are to be refreshed. Valid on z/OS only.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

TYPE Specifies the type of refresh that is to be performed.

AUTHSERV
The list of authorizations held internally by the authorization services component is refreshed.

This is valid only on non-z/OS platforms where it is the default.

CLASSES

WebSphere MQ in-storage ESM (external security manager, for example RACF) profiles are refreshed. The in-storage profiles for the resources being requested are deleted. New entries are created when security checks for them are performed, and are validated when the user next requests access.

You can select specific resource classes for which to perform the security refresh.

This is valid only on z/OS where it is the default.

SSL Refreshes the cached view of the Secure Sockets Layer key repository and allows updates to become effective on successful completion of the command. Also refreshed are the locations of:

- the LDAP servers to be used for Certified Revocation Lists
- the key repository

as well as any cryptographic hardware parameters specified through WebSphere MQ.

RESET CHANNEL:

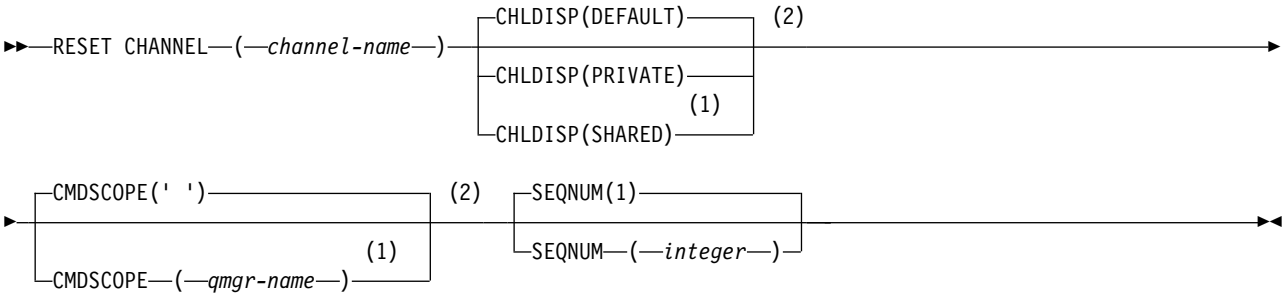
Use the MQSC command RESET CHANNEL to reset the message sequence number for a WebSphere MQ channel with, optionally, a specified sequence number to be used the next time that the channel is started.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for RESET CHANNEL" on page 758

Synonym: RESET CHL

RESET CHANNEL



Notes:

1 Valid only when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

1. On z/OS, the command server and channel initiator must be running.
2. This command can be issued to a channel of any type except SVRCONN and CLNTCONN channels, (including those that have been defined automatically). However, if it is issued to a sender or server channel, then in addition to resetting the value at the end at which the command is issued, the value

at the other (receiver or requester) end is also reset to the same value the next time this channel is initiated (and resynchronized if necessary). Issuing this command on a cluster-sender channel might reset the message sequence number at either end of the channel. However, this is not significant because the sequence numbers are not checked on clustering channels.

3. If the command is issued to a receiver, requester, or cluster-receiver channel, the value at the other end is *not* reset as well; this must be done separately if necessary.
4. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager's repository.
5. If the message is non-persistent, and the RESET CHANNEL command is issued to the sender channel, reset data is sent and flows every time the channel starts.

Parameter descriptions for RESET CHANNEL

(channel-name)
The name of the channel to be reset. This is required.

CHLDISP
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED

If this parameter is omitted, then the DEFAULT value applies. This is taken from the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is **not** related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table:

Table 89. CHLDISP and CMDSCOPE for RESET CHANNEL

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)
PRIVATE	Reset private channel on the local queue manager	Reset private channel on the named queue manager
SHARED	Reset a shared channel on all active queue managers. This might automatically generate a command using CMDSCOPE and send it to the appropriate queue managers. If there is no definition for the channel on the queue managers to which the command is sent, or if the definition is unsuitable for the command, the action fails there. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' '
The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

SEQNUM(*integer***)**

The new message sequence number, which must be in the range 1 through 999 999 999. This is optional.

RESET CLUSTER:

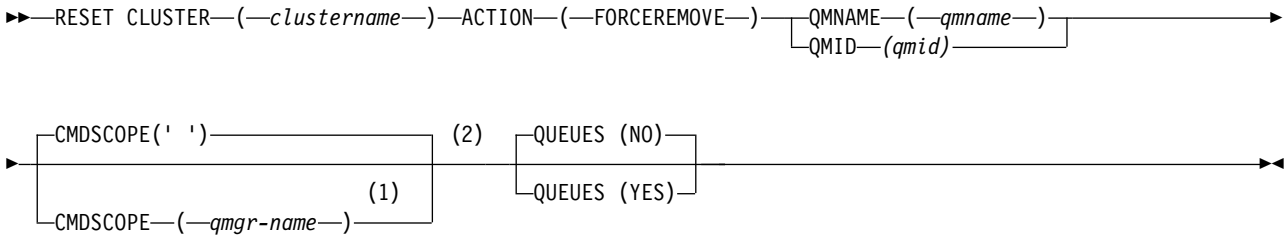
Use the MQSC command **RESET CLUSTER** to perform special operations on clusters.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes for RESET CLUSTER" on page 760
- "Parameter descriptions for RESET CLUSTER" on page 760

Synonym: None

RESET CLUSTER



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for RESET CLUSTER

1. On UNIX systems, the command is valid only on AIX, HP-UX, Linux, and Solaris.
2. On z/OS, the command fails if the channel initiator has not been started.
3. On z/OS, any errors are reported to the console on the system where the channel initiator is running; they are not reported to the system that issued the command.
4. To avoid any ambiguity, it is preferable to use QMID rather than QMNAME. The queue manager identifier can be found by commands such as DISPLAY QMGR and DISPLAY CLUSQMGR.

   If QMNAME is used, and there is more than one queue manager in the cluster with that name, the command is not actioned.
5. If you use characters other than those listed in Rules for naming IBM WebSphere MQ objects in your object or variable names, for example in QMID, you must enclose the name in quotation marks.
6. If you remove a queue manager from a cluster using this command, you can rejoin it to the cluster by issuing a **REFRESH CLUSTER** command. Wait at least 10 seconds before issuing a **REFRESH CLUSTER** command, because the repository ignores any attempt to rejoin the cluster within 10 seconds of a **RESET CLUSTER** command. If the queue manager is in a publish/subscribe cluster, you then need to issue the REFRESH QMGR TYPE(PROXYSUB) command to reinstate any required proxy subscriptions. See REFRESH CLUSTER considerations for publish/subscribe clusters.

   Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

Parameter descriptions for RESET CLUSTER

(clustername)
The name of the cluster to be reset. This is required.

ACTION(FORCEREMOVE)

Requests that the queue manager is forcibly removed from the cluster. This might be needed to ensure correct cleanup after a queue manager has been deleted.
This action can be requested only by a repository queue manager.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

'' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

QMID(*qmid***)**

The identifier of the queue manager to be forcibly removed.

QMNAME(*qmname***)**

The name of the queue manager to be forcibly removed.

QUEUES

Specifies whether cluster queues owned by the queue manager being force removed are removed from the cluster.

NO Cluster queues owned by the queue manager being force removed are not removed from the cluster. This is the default.

YES Cluster queues owned by the queue manager being force removed are removed from the cluster in addition to the cluster queue manager itself. The cluster queues are removed even if the cluster queue manager is not visible in the cluster, perhaps because it was previously force removed without the QUEUES option.

On z/OS, **N** and **Y** are accepted synonyms of **NO** and **YES**.

RESET QMGR:

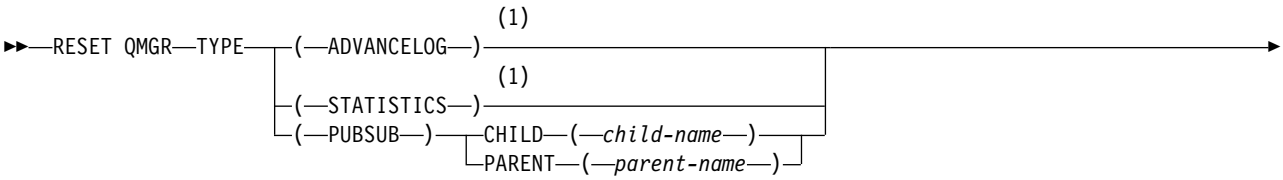
Use the MQSC command RESET QMGR as part of your backup and recovery procedures.

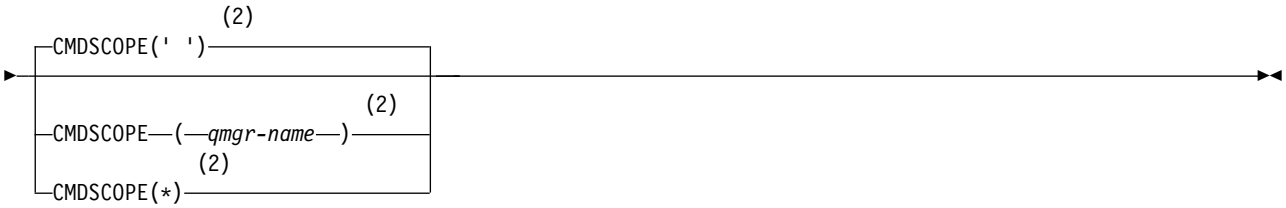
UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes for RESET QMGR" on page 762
- "Parameter descriptions for RESET QMGR" on page 762

Synonym: None

RESET QMGR





Notes:

1 Not valid on z/OS.

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group

Usage notes for RESET QMGR

You can use this command to request that the queue manager starts writing to a new log extent, making the previous log extent available for backup. See Updating a backup queue manager. Alternatively, you can use this command to request that the queue manager ends the current statistics collection period and writes the collected statistics. You can also use this command to forcibly remove a publish/subscribe hierarchical connection for which this queue manager is nominated as either the parent or the child in the hierarchical connection.

1. The queue manager might refuse a request to advance the recovery log, if advancing the recovery log would cause the queue manager to become short of space in the active log.
2. You are unlikely to use RESET QMGR TYPE(PUBSUB) other than in exceptional circumstances. Typically the child queue manager uses ALTER QMGR PARENT(' ') to remove the hierarchical connection.

   When you need to disconnect from a child or parent queue manager with which the queue manager has become unable to communicate, you must issue the RESET QMGR TYPE (PUBSUB) command from a queue manager. When using this command, the remote queue manager is not informed of the canceled connection. It might, therefore, be necessary to issue the ALTER QMGR PARENT(' ') command at the remote queue manager. If the child queue manager is not manually disconnected, it is forcibly disconnected and the parent status is set to REFUSED.

   If you are resetting the parent relationship, issue the ALTER QMGR PARENT(' ') command, otherwise the queue manager attempts to re-establish the connection when the publish/subscribe capability of the queue manager is later enabled.

Parameter descriptions for RESET QMGR

TYPE

ADVANCELOG
Requests that the queue manager starts writing to a new log extent, making the previous log extent available for backup. See Updating a backup queue manager. This command is accepted only if the queue manager is configured to use linear logging.

STATISTICS
Requests that the queue manager ends the current statistics collection period and writes the collected statistics.

PUBSUB
Requests that the queue manager cancels the indicated publish/subscribe hierarchical connection. This value requires that one of the CHILD or PARENT attributes is specified:

CHILD
The name of the child queue manager for which the hierarchical connection is to be forcibly canceled. This attribute is used only with TYPE(PUBSUB). It cannot be used together with PARENT.

PARENT
The name of a parent queue manager for which the hierarchical connection is to be forcibly canceled. This attribute is used only with TYPE(PUBSUB). It cannot be used together with CHILD.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

CMDSCOPE must be blank, or the local queue manager, if QSGDISP is set to GROUP.

' '
The command is executed on the queue manager on which it was entered. This value is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name other than the queue manager on which it was entered, only if you are using a shared queue environment and if the command server is enabled.

*
The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of setting this value is the same as entering the command on every queue manager in the queue-sharing group.

RESOLVE CHANNEL:

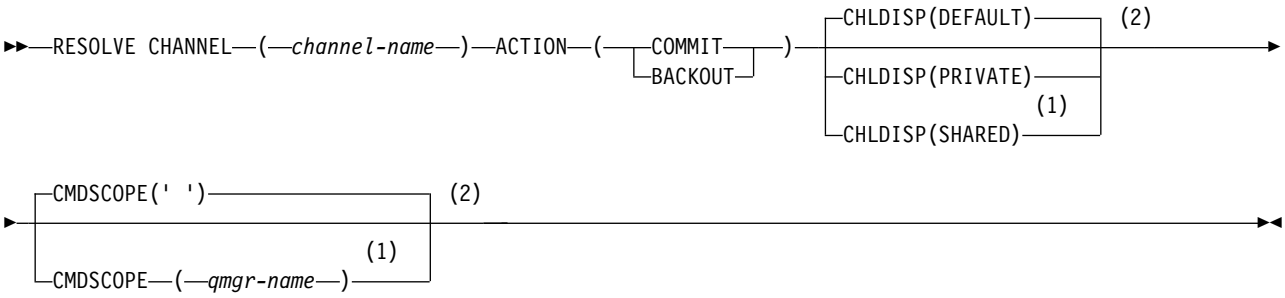
Use the MQSC command RESOLVE CHANNEL to request a channel to commit or back out in-doubt messages.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes for RESOLVE CHANNEL" on page 764
- "Parameter descriptions for RESOLVE CHANNEL" on page 764

Synonym: RESOLVE CHL (RES CHL on z/OS)

RESOLVE CHANNEL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for RESOLVE CHANNEL

1. This command is used when the other end of a link fails during the confirmation period, and for some reason it is not possible to reestablish the connection.
2. In this situation the sending end remains in doubt as to whether the messages were received. Any outstanding units of work must be resolved by being backed out or committed.
3. If the resolution specified is not the same as the resolution at the receiving end, messages can be lost or duplicated.
4. On z/OS, the command server and the channel initiator must be running.
5. This command can be used only for sender (SDR), server (SVR), and cluster-sender (CLUSSDR) channels (including those that have been defined automatically).
6. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager's repository.
7. If the resolution specified is not the same as the resolution at the receiving end, messages can be lost or duplicated.

Parameter descriptions for RESOLVE CHANNEL

(channel-name)
The name of the channel for which in-doubt messages are to be resolved. This is required.

ACTION
Specifies whether to commit or back out the in-doubt messages (this is required):

COMMIT
The messages are committed, that is, they are deleted from the transmission queue

BACKOUT
The messages are backed out, that is, they are restored to the transmission queue

CHLDISP
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED

If this parameter is omitted, then the DEFAULT value applies. This is taken from the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is **not** related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table:

Table 90. CHLDISP and CMDSCOPE for RESOLVE CHANNEL

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)
PRIVATE	Resolve private channel on the local queue manager	Resolve private channel on the named queue manager
SHARED	Resolve a shared channel on all active queue managers. This might automatically generate a command using CMDSCOPE and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

RESUME QMGR:

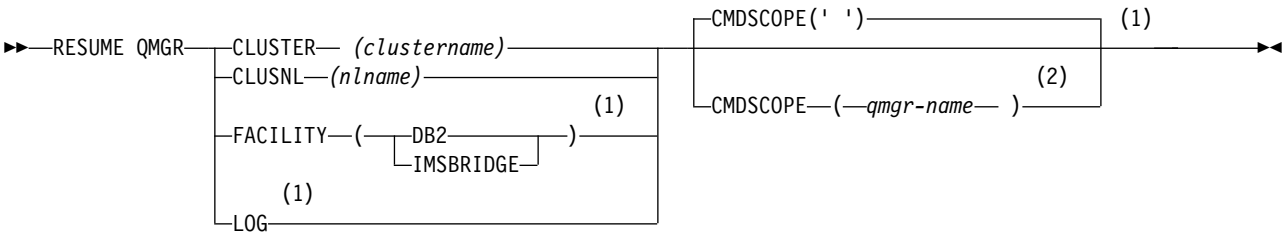
Use the MQSC command RESUME QMGR to inform other queue managers in a cluster that the local queue manager is available again for processing and can be sent messages. It reverses the action of the SUSPEND QMGR command.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes" on page 766
- "Parameter descriptions for RESUME QMGR" on page 766

Synonym: None

RESUME QMGR



Notes:

1 Valid only on z/OS.

2 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

Usage notes

1. On UNIX systems, the command is valid only on AIX, HP-UX, Linux, and Solaris.
2. On z/OS, if you define CLUSTER or CLUSNL:
   a. The command fails if the channel initiator has not been started.
   b. Any errors are reported to the console on the system where the channel initiator is running; they are not reported to the system that issued the command.
3. On z/OS, you cannot issue RESUME QMGR CLUSTER *(clustername)* or RESUME QMGR FACILITY commands from CSQINP2.
4. This command, with the CLUSTER and CLUSNL parameters, is **not** available on the reduced function form of WebSphere MQ for z/OS supplied with WebSphere Application Server.
5. On z/OS, the SUSPEND QMGR and RESUME QMGR commands are supported through the console only. However, all the other SUSPEND and RESUME commands are supported through the console and command server.

Parameter descriptions for RESUME QMGR

CLUSTER*(clustername)*
The name of the cluster for which availability is to be resumed.

CLUSNL*(nlname)*
The name of the namelist specifying a list of clusters for which availability is to be resumed.

FACILITY
Specifies the facility to which connection is to be re-established.

DB2® Re-establishes connection to Db2.

IMSBRIDGE
Resumes normal IMS Bridge activity.

This parameter is only valid on z/OS.

LOG Resumes logging and update activity for the queue manager that was suspended by a previous SUSPEND QMGR command. Valid on z/OS only. If LOG is specified, the command can be issued only from the z/OS console.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

''
The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

SET AUTHREC:

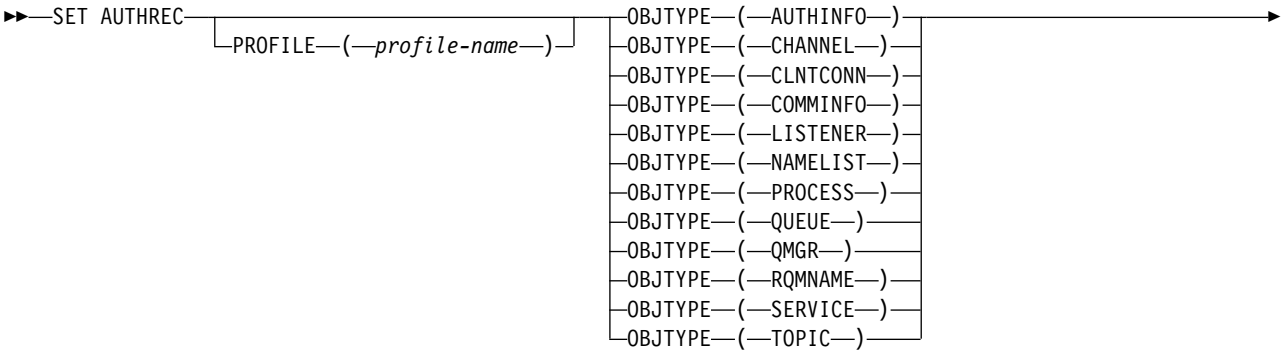
Use the MQSC command SET AUTHREC to set authority records associated with a profile name.

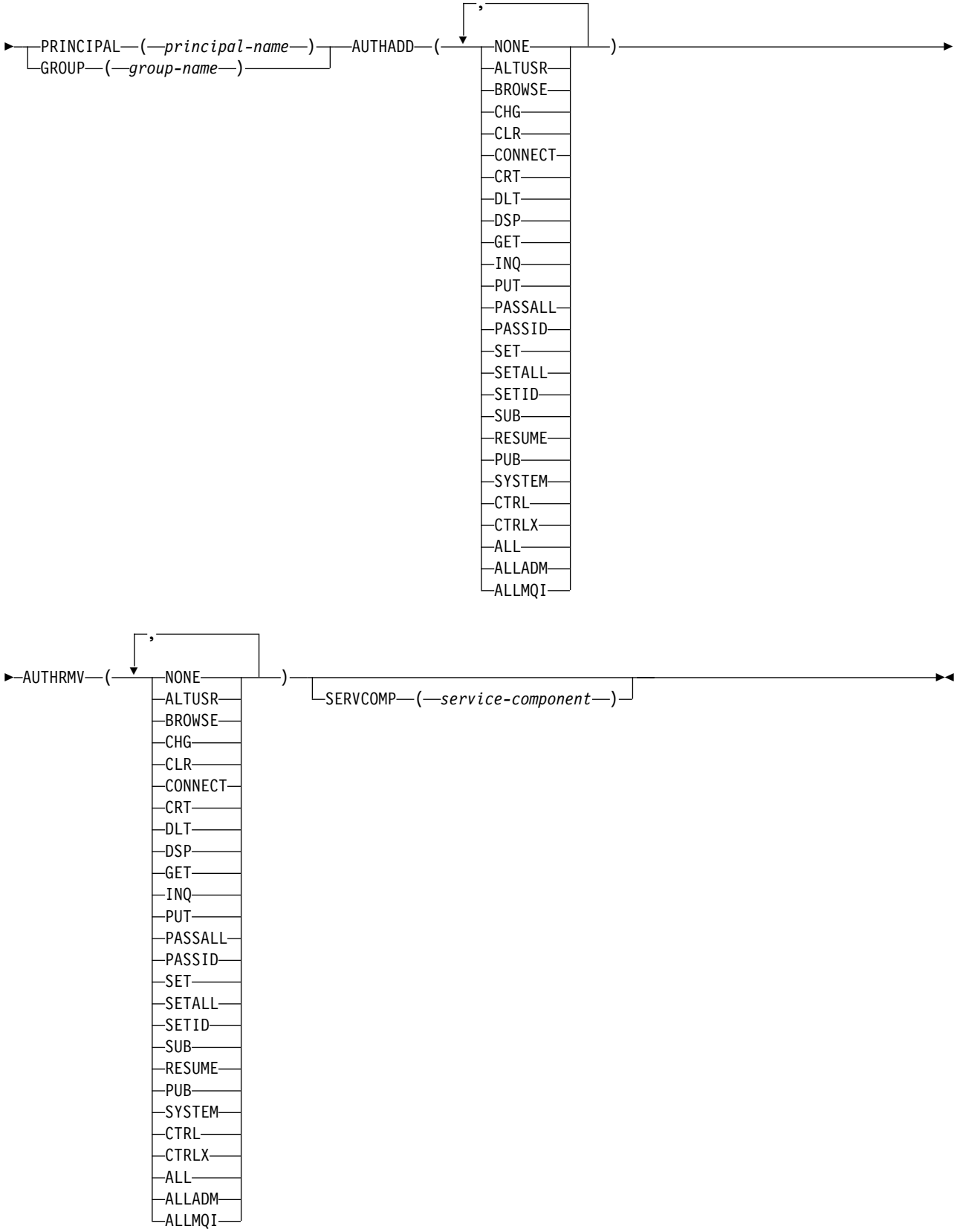
UNIX and Linux	Windows
✓	✓

- - Syntax diagram
- "Parameter descriptions" on page 769
- "Usage notes" on page 772

See "**setmqaut**" on page 222 for more information on the options that you can select.

SET AUTHREC




PRINCIPAL
principal-name
GROUP
group-name
AUTHADD
NONE
ALTUSR
BROWSE
CHG
CLR
CONNECT
CRT
DLT
DSP
GET
INQ
PUT
PASSALL
PASSID
SET
SETALL
SETID
SUB
RESUME
PUB
SYSTEM
CTRL
CTRLX
ALL
ALLADM
ALLMQI
AUTHRMV
NONE
ALTUSR
BROWSE
CHG
CLR
CONNECT
CRT
DLT
DSP
GET
INQ
PUT
PASSALL
PASSID
SET
SETALL
SETID
SUB
RESUME
PUB
SYSTEM
CTRL
CTRLX
ALL
ALLADM
ALLMQI
SERVCOMP
service-component

Parameter descriptions

PROFILE(*profile-name*)
The name of the object or generic profile for which to display the authority records. This parameter is required unless the **OBJTYPE** parameter is QMGR, in which case it can be omitted.

See ../com.ibm.mq.sec.doc/q013500_.dita for more information on generic profiles and wildcard characters.

OBJTYPE
The type of object referred to by the profile. Specify one of the following values:

AUTHINFO
Authentication information record

CHANNEL
Channel

CLNTCONN
Client connection channel

COMMINFO
Communication information object

LISTENER
Listener

NAMELIST
Namelist

PROCESS
Process

QUEUE
Queue

QMGR
Queue manager

RQMNAME
Remote queue manager

SERVICE
Service

TOPIC
Topic

PRINCIPAL(*principal-name*)
A principal name. This is the name of a user for whom to set authority records for the specified profile. On IBM WebSphere MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.

You must specify either PRINCIPAL or GROUP.

GROUP(*group-name*)
A group name. This is the name of the user group for which to set authority records for the specified profile. You can specify one name only and it must be the name of an existing user group.

For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

You must specify either PRINCIPAL or GROUP.

AUTHADD

A list of authorizations to add in the authority records. Specify any combination of the following values:

NONE
No authorization

ALTUSR
Specify an alternative user ID on an MQI call

BROWSE
Retrieve a message from a queue by issuing an **MQGET** call with the BROWSE option

CHG Change the attributes of the specified object, using the appropriate command set

CLR Clear a queue or a topic

CONNECT
Connect an application to a queue manager by issuing an **MQCONN** call

CRT Create objects of the specified type using the appropriate command set

DLT Delete the specified object using the appropriate command set

DSP Display the attributes of the specified object using the appropriate command set

GET Retrieve a message from a queue by issuing an **MQGET** call

INQ Make an inquiry on a specific queue by issuing an **MQINQ** call

PUT Put a message on a specific queue by issuing an **MQPUT** call

PASSALL
Pass all context

PASSID
Pass the identity context

SET Set attributes on a queue by issuing an **MQSET** call

SETALL
Set all context on a queue

SETID
Set the identity context on a queue

SUB Create, alter, or resume a subscription to a topic using the **MQSUB** call

RESUME
Resume a subscription using the MQSUB call

PUB Publish a message on a topic using the **MQPUT** call

SYSTEM
Use queue manager for internal system operations

CTRL Start and stop the specified channel, listener, or service, and ping the specified channel

CTRLX
Reset or resolve the specified channel

ALL Use all operations relevant to the object

all authority is equivalent to the union of the authorities alladm, allmqi, and system appropriate to the object type.

ALLADM
Perform all administration operations relevant to the object

ALLMQI
Use all MQI calls relevant to the object

AUTHRMV
A list of authorizations to remove from the authority records. Specify any combination of the following values:

NONE
No authorization

ALTUSR
Specify an alternative user ID on an MQI call

BROWSE
Retrieve a message from a queue by issuing an **MQGET** call with the BROWSE option

CHG Change the attributes of the specified object, using the appropriate command set

CLR Clear a queue or a topic

CONNECT
Connect an application to a queue manager by issuing an **MQCONN** call

CRT Create objects of the specified type using the appropriate command set

DLT Delete the specified object using the appropriate command set

DSP Display the attributes of the specified object using the appropriate command set

GET Retrieve a message from a queue by issuing an **MQGET** call

INQ Make an inquiry on a specific queue by issuing an **MQINQ** call

PUT Put a message on a specific queue by issuing an **MQPUT** call

PASSALL
Pass all context

PASSID
Pass the identity context

SET Set attributes on a queue by issuing an **MQSET** call

SETALL
Set all context on a queue

SETID
Set the identity context on a queue

SUB Create, alter, or resume a subscription to a topic using the **MQSUB** call

RESUME
Resume a subscription using the MQSUB call

PUB Publish a message on a topic using the **MQPUT** call

SYSTEM
Use queue manager for internal system operations

CTRL Start and stop the specified channel, listener, or service, and ping the specified channel

CTRLX
Reset or resolve the specified channel

ALL Use all operations relevant to the object

all authority is equivalent to the union of the authorities alladm, allmqi, and system appropriate to the object type.

ALLADM
Perform all administration operations relevant to the object

ALLMQI
Use all MQI calls relevant to the object

SERVCOMP(*service-component*)
The name of the authorization service for which information is to be set.

If you specify this parameter, it specifies the name of the authorization service to which the authorizations apply. If you omit this parameter, the authority record is set using the registered authorization services in turn in accordance with the rules for chaining authorization services.

Usage notes

The list of authorizations to add and the list of authorizations to remove must not overlap. For example, you cannot add display authority and remove display authority with the same command. This rule applies even if the authorities are expressed using different options. For example, the following command fails because DSP authority overlaps with ALLADM authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP) AUTHRMV(ALLADM)
```

The exception to this overlap behavior is with the ALL authority. The following command first adds ALL authorities then removes the SETID authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(ALL) AUTHRMV(SETID)
```

The following command first removes ALL authorities then adds the DSP authority:

```
SET AUTHREC PROFILE(*) OBJTYPE(QUEUE) PRINCIPAL(PRINC01) AUTHADD(DSP) AUTHRMV(ALL)
```

Regardless of the order in which they are provided on the command, the ALL are processed first.

SET CHLAUTH:

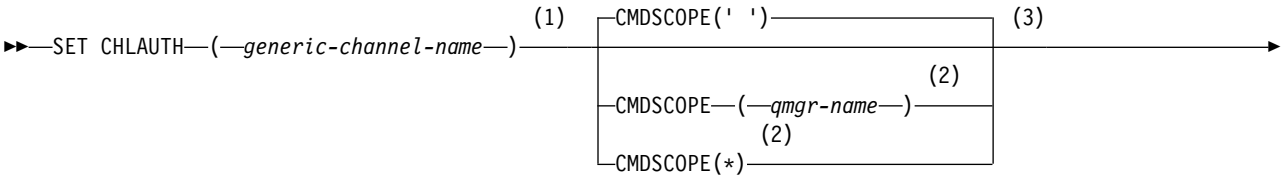
Use the MQSC command SET CHLAUTH to create or modify a channel authentication record.

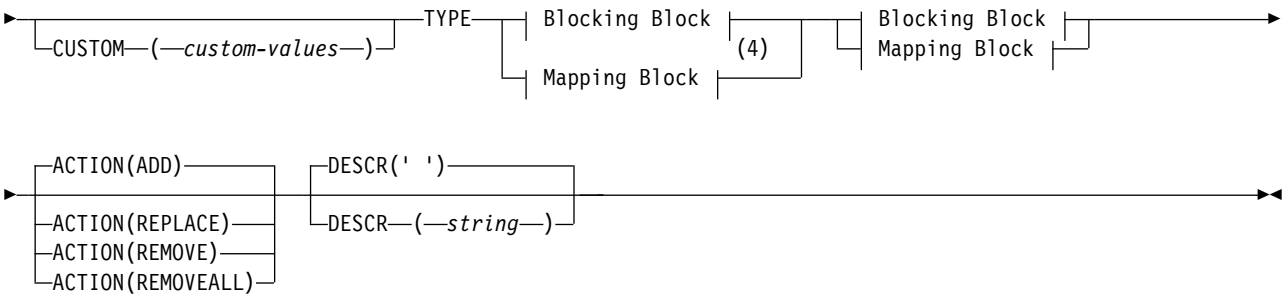
UNIX and Linux	Windows
✓	✓

For an explanation of the symbols in the z/OS column, see .

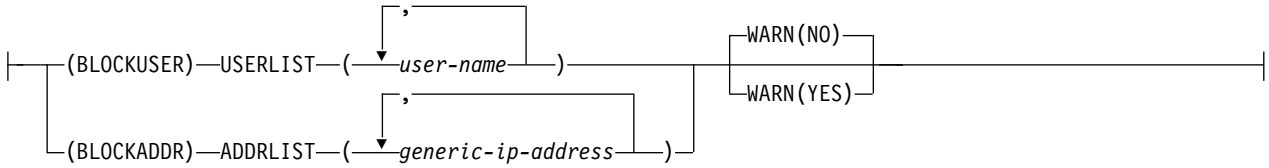
- Syntax diagram
- Usage notes
- Parameters

SET CHLAUTH

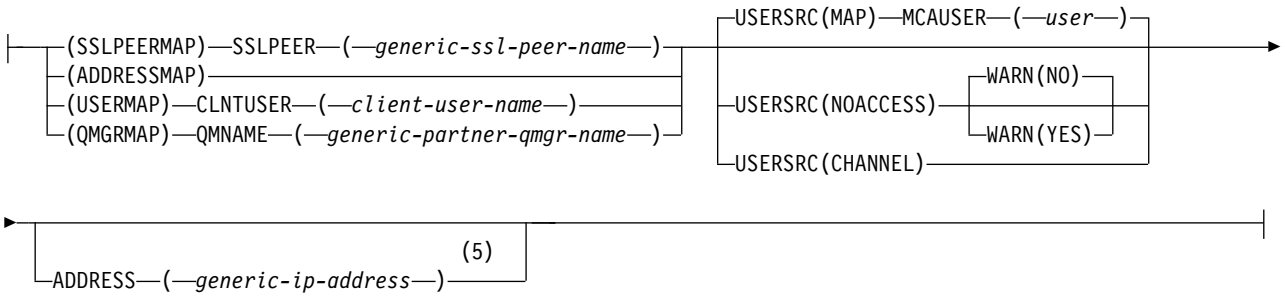




Blocking Block:



Mapping Block:



Notes:

1 The generic channel name must be '*' when TYPE is BLOCKADDR

2 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

4 Select the appropriate value for TYPE, depending upon the option that you select from the two types of block.

5 Mandatory when TYPE is ADDRESSMAP

Usage notes

The following table shows which parameters are valid for each value of **ACTION**:

	Action		
Parameter	**ADD or REPLACE**	**REMOVE**	**REMOVEALL**
CHLAUTH	✓	✓	✓
TYPE	✓	✓	✓
CMDSCOPE	✓	✓	✓
ACTION	✓	✓	✓
ADDRESS	✓	✓	
ADDRLIST	✓	✓	
CLNTUSER	✓	✓	
MCAUSER	✓		
QMNAME	✓	✓	
SSLPEER	✓	✓	
USERLIST	✓	✓	
USERSRC	✓		
WARN	✓		
DESCR	✓		

Parameters

generic-channel-name
The name of the channel or set of channels for which you are setting channel authentication configuration. You can use one or more asterisks (*), in any position, as wildcards to specify a set of channels. If you set **TYPE** to BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names. On z/OS the generic-channel-name must be in quotes if it contains an asterisk.

TYPE
The **TYPE** parameter must follow the **generic-channel-name** parameter.

The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER. This parameter is required. The following values can be used:

BLOCKUSER
This channel authentication record prevents a specified user or users from connecting. The BLOCKUSER parameter must be accompanied by a USERLIST.

BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses. The BLOCKADDR parameter must be accompanied by an ADDRLIST. BLOCKADDR operates at the listener before the channel name is known.

SSLPEERMAP
This channel authentication record maps SSL or TLS Distinguished Names (DNs) to MCAUSER values. The SSLPEERMAP parameter must be accompanied by an SSLPEER.

ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values. The ADDRESSMAP parameter must be accompanied by an ADDRESS. ADDRESSMAP operates at the channel.

USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values. The USERMAP parameter must be accompanied by a CLNTUSER.

QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values. The QMGRMAP parameter must be accompanied by a QMNAME.

ACTION
The action to perform on the channel authentication record. The following values are valid:

ADD Add the specified configuration to a channel authentication record. This is the default value.

For types SSLPEERMAP, ADDRESSMAP, USERMAP and QMGRMAP, if the specified configuration exists, the command fails.

For types BLOCKUSER and BLOCKADDR, the configuration is added to the list.

REPLACE
Replace the current configuration of a channel authentication record.

For types SSLPEERMAP, ADDRESSMAP, USERMAP and QMGRMAP, if the specified configuration exists, it is replaced with the new configuration. If it does not exist it is added.

For types BLOCKUSER and BLOCKADDR, the configuration specified replaces the current list, even if the current list is empty. If you replace the current list with an empty list, this acts like REMOVEALL.

REMOVE
Remove the specified configuration from the channel authentication records. If the configuration does not exist the command fails. If you remove the last entry from a list, this acts like REMOVEALL.

REMOVEALL
Remove all members of the list and thus the whole record (for BLOCKADDR and BLOCKUSER) or all previously defined mappings (for ADDRESSMAP, SSLPEERMAP, QMGRMAP and USERMAP) from the channel authentication records. This option cannot be combined with specific values supplied in **ADDRLIST**, **USERLIST**, **ADDRESS**, **SSLPEER**, **QMNAME** or **CLNTUSER**. If the specified type has no current configuration the command still succeeds.

ADDRESS
The filter to be used to compare with the IP address of the partner queue manager or client at the other end of the channel.

This parameter is mandatory with **TYPE(ADDRESSMAP)**

This parameter is also valid when **TYPE** is SSLPEERMAP, USERMAP, or QMGRMAP and **ACTION** is ADD, REPLACE, or REMOVE. You can define more than one channel authentication object with the same main identity, for example the same SSL peer name, with different addresses. However, you cannot define channel authentication records with overlapping address ranges for the same main identity. See “Generic IP addresses” on page 778 for more information about filtering IP addresses.

If the address is generic then it must be in quotes.

ADDRLIST
A list of up to 256 generic IP addresses which are banned from accessing this queue manager on any channel. This parameter is only valid with TYPE(BLOCKADDR). See “Generic IP addresses” on page 778 for more information about filtering IP addresses.

If the address is generic then it must be in quotes.

CLNTUSER
The client asserted user ID to be mapped to a new user ID or blocked.

This parameter is valid only with **TYPE(USERMAP)**.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is run when the queue manager is a member of a queue-sharing group.

' ' The command is run on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is run on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is run on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect is the same as entering the command on every queue manager in the queue-sharing group.

CUSTOM
Reserved for future use.

DESCR
Provides descriptive information about the channel authentication record, which is displayed when you issue the DISPLAY CHLAUTH command. It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

MCAUSER
The user identifier to be used when the inbound connection matches the SSL or TLS DN, IP address, client asserted user ID or remote queue manager name supplied.

This parameter is mandatory with **USERSRC(MAP)** and is valid when **TYPE** is SSLPEERMAP, ADDRESSMAP, USERMAP, or QMGRMAP.

This parameter can only be used when **ACTION** is ADD or REPLACE.

QMNAME
The name of the remote partner queue manager, or pattern that matches a set of queue manager names, to be mapped to a user ID or blocked.

This parameter is valid only with **TYPE(QMGRMAP)**.

If the queue manager name is generic then it must be in quotes.

SSLPEER

The filter to use to compare with the Subject Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel.

The **SSLPEER** filter is specified in the standard form used to specify a Distinguished Name. See WebSphere MQ rules for SSLPEER values for details.

The maximum length of the parameter is 1024 bytes.

USERLIST

A list of up to 100 user IDs which are banned from use of this channel or set of channels. Use the special value *MQADMIN to mean privileged or administrative users. The definition of this value depends on the operating system, as follows:

- On Windows, all members of the mqm group, the Administrators group and SYSTEM.
- On UNIX and Linux, all members of the mqm group.
- On IBM i, the profiles (users) qmqm and qmqmadm and all members of the qmqmadm group, and any user defined with the *ALLOBJ special setting.
- On z/OS, the user ID that the channel initiator and queue manager address spaces are running under.

For more information about privileged users, see Privileged users .

This parameter is only valid with **TYPE(BLOCKUSER)**.

USERSRC

The source of the user ID to be used for MCAUSER at run time. The following values are valid:

MAP Inbound connections that match this mapping use the user ID specified in the **MCAUSER** attribute. This is the default value.

NOACCESS
Inbound connections that match this mapping have no access to the queue manager and the channel ends immediately.

CHANNEL
Inbound connections that match this mapping use the flowed user ID or any user defined on the channel object in the MCAUSER field.

Note that WARN and USERSRC(CHANNEL), or USERSRC(MAP) are incompatible. This is because channel access is never blocked in these cases, so there is never a reason to generate a warning.

WARN

Indicates whether this record operates in warning mode.

NO This record does not operate in warning mode. Any inbound connection that matches this record is blocked. This is the default value.

YES This record operates in warning mode. Any inbound connection that matches this record and would therefore be blocked is allowed access. An error message is written and, if channel events are configured, a channel event message is created showing the details of what would have been blocked, see Channel Blocked. The connection is allowed to continue. An attempt is made to find another record that is set to WARN(NO) to set the credentials for the inbound channel.

Related information:

Channel authentication records

Securing remote connectivity to the queue manager

Generic IP addresses:

In the various commands that create and display channel authentication records, you can specify certain parameters as either a single IP address or a pattern to match a set of IP addresses.

When you create a channel authentication record, using the MQSC command SET CHLAUTH or the PCF command Set Channel Authentication Record , you can specify a generic IP address in various contexts. You can also specify a generic IP address in the filter condition when you display a channel authentication record using the commands DISPLAY CHLAUTH or Inquire Channel Authentication Records .

You can specify the address in any of the following ways:

- a single IPv4 address, such as 192.0.2.0
- a pattern based on an IPv4 address, including an asterisk (*) as a wildcard. The wildcard represents one or more parts of the address, depending on context. For example, the following are all valid values:
  - 192.0.2.*
  - 192.0.*
  - 192.0.*.2
  - 192.*.2
  - *
- a pattern based on an IPv4 address, including a hyphen (-) to indicate a range, for example 192.0.2.1-8
- a pattern based on an IPv4 address, including both an asterisk and a hyphen, for example 192.0.*.1-8
- a single IPv6 address, such as 2001:DB8:0:0:0:0:0:0
- a pattern based on an IPv6 address including an asterisk (*) as a wildcard. The wildcard represents one or more parts of the address, depending on context. For example, the following are all valid values:
  - 2001:DB8:0:0:0:0:0:*
  - 2001:DB8:0:0:0:*
  - 2001:DB8:0:0:0:*:0:1
  - 2001:*:1
  - *
- a pattern based on an IPv6 address, including a hyphen (-) to indicate a range, for example 2001:DB8:0:0:0:0:0:0-8
- a pattern based on an IPv6 address, including both an asterisk and a hyphen, for example 2001:DB8:0:0:0:*:0:0-8

If your system supports both IPv4 and IPv6, you can use either address format. IBM WebSphere MQ recognizes IPv4 mapped addresses in IPv6.

Certain patterns are invalid:

- A pattern cannot have fewer than the required number of parts, unless the pattern ends with a single trailing asterisk. For example 192.0.2 is invalid, but 192.0.2.* is valid.
- A trailing asterisk must be separated from the rest of the address by the appropriate part separator (a dot (.) for IPv4, a colon (:) for IPv6). For example, 192.0* is not valid because the asterisk is not in a part of its own.

- A pattern may contain additional asterisks provided that no asterisk is adjacent to the trailing asterisk. For example, 192.*.2.* is valid, but 192.0.*.* is not valid.
- An IPv6 address pattern cannot contain a double colon and a trailing asterisk, because the resulting address would be ambiguous. For example, 2001::* could expand to 2001:0000:*, 2001:0000:0000:* and so on

Related information:

Mapping an IP address to an MCAUSER user ID

START CHANNEL:

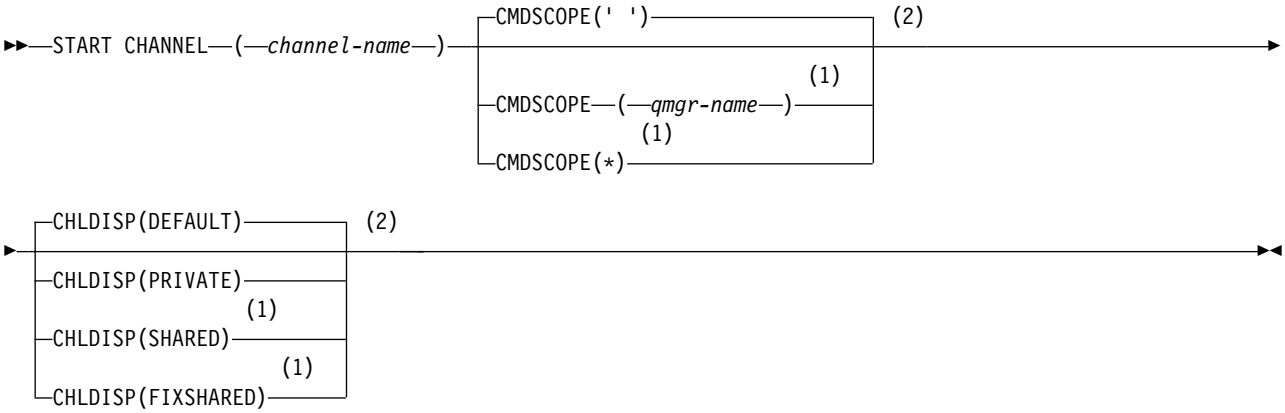
Use the MQSC command START CHANNEL to start a channel.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for START CHANNEL" on page 780

Synonym: STA CHL

START CHANNEL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes

1. On z/OS, the command server and the channel initiator must be running.
2. This command can be issued to a channel of any type except CLNTCONN channels (including those that have been defined automatically). If, however, it is issued to a receiver (RCVR), server-connection (SVRCONN) or cluster-receiver (CLUSRCVR) channel, the only action is to enable the channel, not to start it.
3. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager's repository.

Parameter descriptions for START CHANNEL

(channel-name)
The name of the channel definition to be started. This is required for all channel types. The name must be that of an existing channel.

CHLDISP
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED
- FIXSHARED

If this parameter is omitted, then the DEFAULT value applies. This is taken from the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is not related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On every active queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table:

Table 91. CHLDISP and CMDSCOPE for START CHANNEL

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	Start as a private channel on the local queue manager	Start as a private channel on the named queue manager	Start as a private channel on all active queue managers

Table 91. CHLDISP and CMDSCOPE for START CHANNEL (continued)

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
SHARED	For a shared SDR, RQSTR, and SVR channel, start as a shared channel on the most suitable queue manager in the group. For a shared RCVR and SVRCONN channel, start the channel as a shared channel on all active queue managers. For a shared CLUSSDR or CLUSRCVR channel, this option is not permitted. This might automatically generate a command using CMDSCOPE and send it to the appropriate queue managers. If there is no definition for the channel on the queue managers to which the command is sent, or if the definition is unsuitable for the command, the action fails there. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted
FIXSHARED	For a shared SDR, RQSTR, and SVR channel, with a nonblank CONNAME, start as a shared channel on the local queue manager. For all other types, this option is not permitted.	For a shared SDR, RQSTR, and SVR with a nonblank CONNAME, start as a shared channel on the named queue manager. For all other types, this option is not permitted.	Not permitted

Channels started with CHLDISP(FIXSHARED) are tied to the specific queue manager; if the channel initiator on that queue manager stops for any reason, the channels are not recovered by another queue manager in the group.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

* The command is executed on the local queue manager and is also passed to every active

queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

This option is not permitted if CHLDISP is FIXSHARED.

START CHANNEL (MQTT):

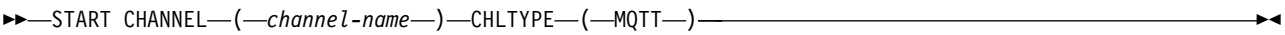
Use the MQSC command START CHANNEL to start a IBM WebSphere MQ Telemetry channel.

IBM i	UNIX and Linux	Windows	z/OS
	✓	✓	

The START CHANNEL (MQTT) command is only valid for IBM WebSphere MQ Telemetry channels. Supported platforms for IBM WebSphere MQ Telemetry are AIX, Linux, Windows.

Synonym: STA CHL

START CHANNEL



Parameter descriptions for START CHANNEL

(channel-name)
The name of the channel definition to be started. The name must be that of an existing channel.

CHLTYPE
Channel type. The value must be MQTT.

START CHINIT:

Use the MQSC command START CHINIT to start a channel initiator.

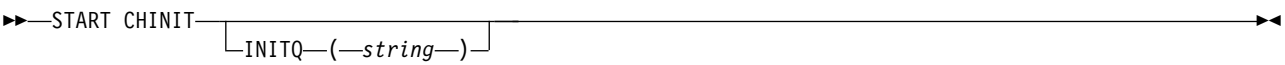
UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes" on page 783
- "Parameter descriptions for START CHINIT" on page 783

Synonym: STA CHI

Syntax diagram

START CHINIT



Usage notes

Parameter descriptions for START CHINIT

INITQ(*string***)**
The name of the initiation queue for the channel initiation process. This is the initiation queue that is specified in the definition of the transmission queue.

On AIX, HP-UX, Linux, IBM i, Solaris, and Windows, you can specify which initiation queue to use; if you do not specify this, SYSTEM.CHANNEL.INITQ is used. On other platforms it must be specified.

START LISTENER:

Use the MQSC command START LISTENER to start a channel listener.

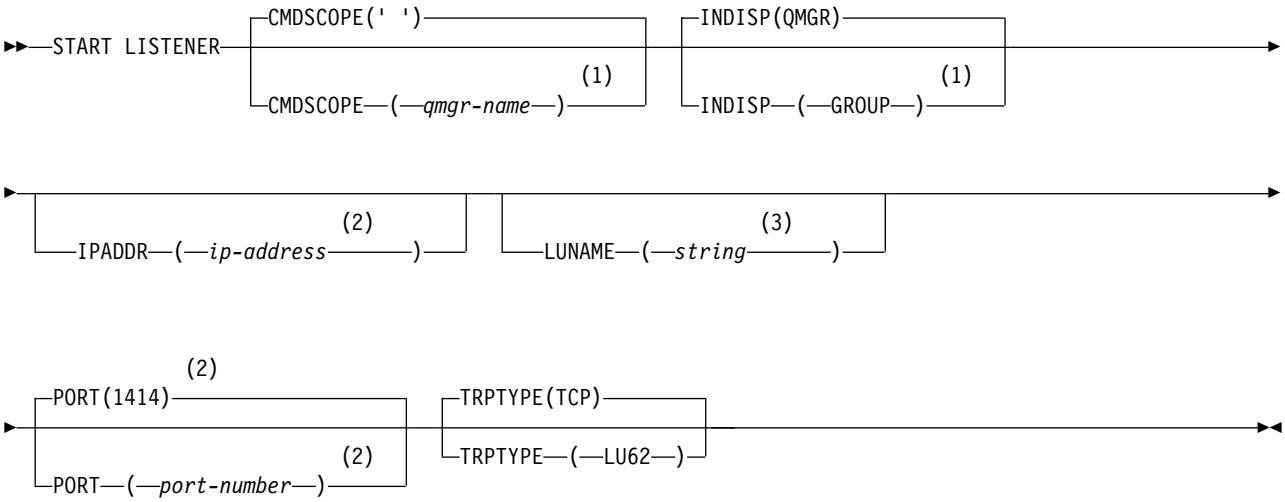
UNIX and Linux	Windows
✓	✓

- Syntax diagram for WebSphere MQ for z/OS
- Syntax diagram for WebSphere MQ on other platforms
- "Usage notes" on page 784
- "Parameter descriptions for START LISTENER" on page 784

Synonym: STA LSTR

WebSphere MQ for z/OS

START LISTENER



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only for TRPTYPE(TCP).

3 Valid only for TRPTYPE(LU62).

WebSphere MQ on other platforms

START LISTENER

Usage notes

1. On z/OS:
   a. The command server and the channel initiator must be running.
   b. If IPADDR is not specified, the listener listens on all available IPv4 and IPv6 addresses.
   c. For TCP/IP, it is possible to listen on multiple addresses and port combinations.
   d. For each START LISTENER for TCP/IP request, the address and port combination is added to the list of combinations upon which the listener is currently listening.
   e. A START LISTENER for TCP/IP request fails if it specifies the same, or a subset or superset of an existing, combination of addresses and ports upon which a TCP/IP listener is currently listening.
   f. If you are starting a listener on a specific address to provide a secure interface with a security product, for example a firewall, it is important to ensure there is no linkage to the other non-secure interfaces in the system.

      You should disable IP forwarding and routing from other non-secure interfaces so that packets arriving at the other interface do not get passed to this specific address.

      Consult the appropriate TCP/IP documentation for information on how to do this.
2. On IBM i, UNIX systems, and Windows, this command is valid only for channels for which the transmission protocol (TRPTYPE) is TCP.

Parameter descriptions for START LISTENER

(*name***)** Name of the listener to be started. If you specify this parameter, you cannot specify any other parameters.

If you do not specify a name (on platforms other than z/OS), the SYSTEM.DEFAULT.LISTENER.TCP is started.

This parameter is not valid on z/OS.

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

INDISP

Specifies the disposition of the inbound transmissions that are to be handled. The possible values are:

QMGR

Listen for transmissions directed to the queue manager. This is the default.

GROUP
Listen for transmissions directed to the queue-sharing group. This is allowed only if there is a shared queue manager environment.

This parameter is valid only on z/OS.

IPADDR
IP address for TCP/IP specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric form. This is valid only if the transmission protocol (TRPTYPE) is TCP/IP.

This parameter is valid only on z/OS.

LUNAME(*string***)**
The symbolic destination name for the logical unit as specified in the APPC side information data set. (This must be the same LU that was specified for the queue manager, using the LUNAME parameter of the ALTER QMGR command.)

This parameter is valid only for channels with a transmission protocol (TRPTYPE) of LU 6.2. A START LISTENER command that specifies TRPTYPE(LU62) must also specify the LUNAME parameter.

This parameter is valid only on z/OS.

PORT(*port-number***)**
Port number for TCP. This is valid only if the transmission protocol (TRPTYPE) is TCP.

This parameter is valid only on z/OS.

TRPTYPE
Transport type to be used. This is optional.

TCP TCP. This is the default if TRPTYPE is not specified.

LU62 SNA LU 6.2.

This parameter is valid only on z/OS.

START SERVICE:

Use the MQSC command START SERVICE to start a service. The identified service definition is started within the queue manager and inherits the environment and security variables of the queue manager.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Parameter descriptions for START SERVICE"

Synonym:

START SERVICE

Parameter descriptions for START SERVICE

(*service-name***)**
The name of the service definition to be started. This is required. The name must that of an existing service on this queue manager.

If the service is already running, and the operating system task is active, an error is returned.

Related information:

Working with services

STOP CHANNEL:

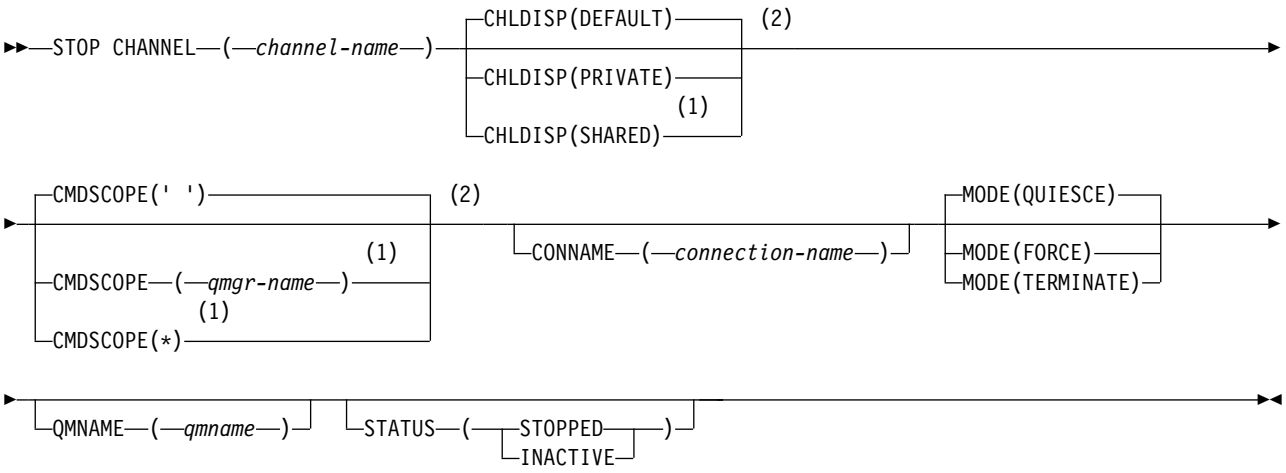
Use the MQSC command STOP CHANNEL to stop a channel.

UNIX and Linux	Windows	
✓	✓	✓

- Syntax diagram
- "Usage notes for STOP CHANNEL"
- "Parameter descriptions for STOP CHANNEL" on page 787

Synonym: STOP CHL

STOP CHANNEL



Notes:

1 Valid only on z/OS when the queue manager is a member of a queue-sharing group.

2 Valid only on z/OS.

Usage notes for STOP CHANNEL

1. If you specify either QMNAME or CONNAME, STATUS must either be INACTIVE or not specified. Do not specify a QMNAME or CONNAME and STATUS(STOPPED). It is not possible to have a channel stopped for one partner but not for others. This sort of function can be provided by a channel security exit. For more information about channel exits, see Channel exit programs.
2. On z/OS, the command server and the channel initiator must be running.
3. Any channels in STOPPED state need to be started manually; they are not started automatically. See Restarting stopped channels for information about restarting stopped channels.
4. This command can be issued to a channel of any type except CLNTCONN channels (including those that have been defined automatically).

5. Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel. If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the channel that was last added to the local queue manager repository.
6. If you issue a `STOP CHANNEL(<channelname>) MODE QUIESCE` command on a server-connection channel with the sharing conversations feature enabled, the IBM WebSphere MQ client infrastructure becomes aware of the stop request in a timely manner; this time is dependent upon the speed of the network. The client application becomes aware of the stop request as a result of issuing a subsequent call to IBM WebSphere MQ.

Parameter descriptions for STOP CHANNEL

(channel-name)
The name of the channel to be stopped. This parameter is required for all channel types.

CHLDISP
This parameter applies to z/OS only and can take the values of:

- DEFAULT
- PRIVATE
- SHARED

If this parameter is omitted, then the DEFAULT value applies. This is taken from the default channel disposition attribute, DEFCDISP, of the channel object.

In conjunction with the various values of the CMDSCOPE parameter, this parameter controls two types of channel:

SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of SHARED.

PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than SHARED.

Note: This disposition is not related to the disposition set by the disposition of the queue-sharing group of the channel definition.

The combination of the CHLDISP and CMDSCOPE parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On every active queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of CHLDISP and CMDSCOPE are summarized in the following table:

Table 92. CHLDISP and CMDSCOPE for STOP CHANNEL

CHLDISP	CMDSCOPE( ) or CMDSCOPE (local-qmgr)	CMDSCOPE (qmgr-name)	CMDSCOPE(*)
PRIVATE	Stop as a private channel on the local queue manager.	Stop as a private channel on the named queue manager	Stop as a private channel on all active queue managers
SHARED	For RCVR and SVRCONN channels, stop as shared channel on all active queue managers. For SDR, RQSTR, and SVR channels, stop as a shared channel on the queue manager where it is running. If the channel is in an inactive state (not running), or if it is in RETRY state because the channel initiator on which it was running has stopped, a STOP request for the channel is issued on the local queue manager. This might automatically generate a command using CMDSCOPE and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is actually run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted

CMDSCOPE

This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

If CHLDISP is set to SHARED, CMDSCOPE must be blank or the local queue manager.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name only if you are using a queue-sharing group environment and if the command server is enabled.

***** The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group. The effect of this is the same as entering the command on every queue manager in the queue-sharing group.

CONNAME*(connection-name)*

Connection name. Only channels matching the specified connection name are stopped

MODE

Specifies whether the current batch is allowed to finish in a controlled manner. This parameter is optional.

QUIESCE

Allows the current batch to finish processing, except on z/OS where the channel stops after the current message has finished processing. (The batch is then ended and no more messages are sent, even if there are messages waiting on the transmission queue.)

For a receiving channel, if there is no batch in progress, the channel waits for either:

- The next batch to start
- The next heartbeat (if heartbeats are being used)

before it stops.

For server-connection channels, allows the current connection to end.

If you issue a `STOP CHANNEL <channelname> MODE (QUIESCE)` command on a server-connection channel with the sharing conversations feature enabled, the IBM WebSphere MQ client infrastructure becomes aware of the stop request in a timely manner; this time is dependent upon the speed of the network. The client application becomes aware of the stop request as a result of issuing a subsequent call to IBM WebSphere MQ.

This is the default.

FORCE

For server-connection channels, breaks the current connection, returning MQRC_CONNECTION_BROKEN.For other channel types, terminates transmission of any current batch. This is likely to result in in-doubt situations.

On IBM WebSphere MQ for z/OS, specifying FORCE interrupts any message reallocation in progress, which might leave BIND_NOT_FIXED messages partially reallocated or out of order.

TERMINATE

On z/OS this is synonymous with FORCE. On other platforms, this parameter terminates transmission of any current batch. This allows the command to actually terminate the channel thread or process.

For server-connection channels, breaks the current connection, returning MQRC_CONNECTION_BROKEN.

On IBM WebSphere MQ for z/OS, specifying TERMINATE interrupts any message reallocation in progress, which might leave BIND_NOT_FIXED messages partially reallocated or out of order.

QMNAME*(qmname)*

Queue manager name. Only channels matching the specified remote queue manager are stopped

STATUS

Specifies the new state of any channels stopped by this command. For details about channels in STOPPED state, and especially SVRCONN channels, see Restarting stopped channels.

STOPPED

The channel is stopped. For a sender or server channel the transmission queue is set to GET(DISABLED) and NOTRIGGER.

This is the default if QMNAME or CONNAME are not specified.

INACTIVE

The channel is inactive.

This is the default if QMNAME or CONNAME are specified.

STOP CHANNEL (MQTT):

Use the MQSC command STOP CHANNEL to stop a IBM WebSphere MQ Telemetry channel.

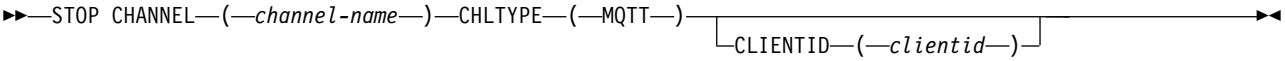
IBM i	UNIX and Linux	Windows	z/OS
	✓	✓	

Note: For the telemetry server, AIX is the only supported UNIX platform.

The STOP CHANNEL (MQTT) command is only valid for IBM WebSphere MQ Telemetry channels.

Synonym: STOP CHL

STOP CHANNEL



Usage notes for STOP CHANNEL

1. Any channels in STOPPED state need to be started manually; they are not started automatically. See Restarting stopped channels for information about restarting stopped channels.

Parameter descriptions for STOP CHANNEL

(channel-name)
The name of the channel to be stopped. This parameter is required for all channel types including MQTT channels.

CHLTYPE
Channel type. Tthe value must be MQTT.

CLIENTID*(string)*
Client identifier. The client identifier is a 23-byte string that identifies a IBM WebSphere MQ Telemetry Transport client. When the STOP CHANNEL command specifies a CLIENTID, only the connection for the specified client identifier is stopped. If the CLIENTID is not specified, all the connections on the channel are stopped.

STOP CONN:

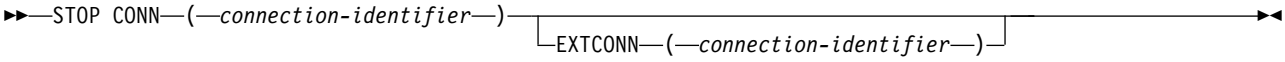
Use the MQSC command STOP CONN to break a connection between an application and the queue manager.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes" on page 791
- "Parameter descriptions for STOP CONN" on page 791

Synonym: STOP CONN

STOP CONN



Usage notes

There might be circumstances in which the queue manager cannot implement this command when the success of this command cannot be guaranteed.

Parameter descriptions for STOP CONN

(*connection-identifier***)**
The identifier of the connection definition for the connection to be broken.

When an application connects to WebSphere MQ, it is given a unique 24-byte connection identifier (ConnectionId). The value of CONN is formed by converting the last eight bytes of the ConnectionId to its 16-character hexadecimal equivalent.

EXTCONN
The value of EXTCONN is based on the first sixteen bytes of the ConnectionId converted to its 32-character hexadecimal equivalent.

Connections are identified by a 24-byte connection identifier. The connection identifier comprises a prefix, which identifies the queue manager, and a suffix which identifies the connection to that queue manager. By default, the prefix is for the queue manager currently being administered, but you can specify a prefix explicitly by using the EXTCONN parameter. Use the CONN parameter to specify the suffix.

When connection identifiers are obtained from other sources, specify the fully qualified connection identifier (both EXTCONN and CONN) to avoid possible problems related to non-unique CONN values.

Related reference:

"DISPLAY CONN" on page 639
Use the MQSC command DISPLAY CONN to display connection information about the applications connected to the queue manager. This is a useful command because it enables you to identify applications with long-running units of work.

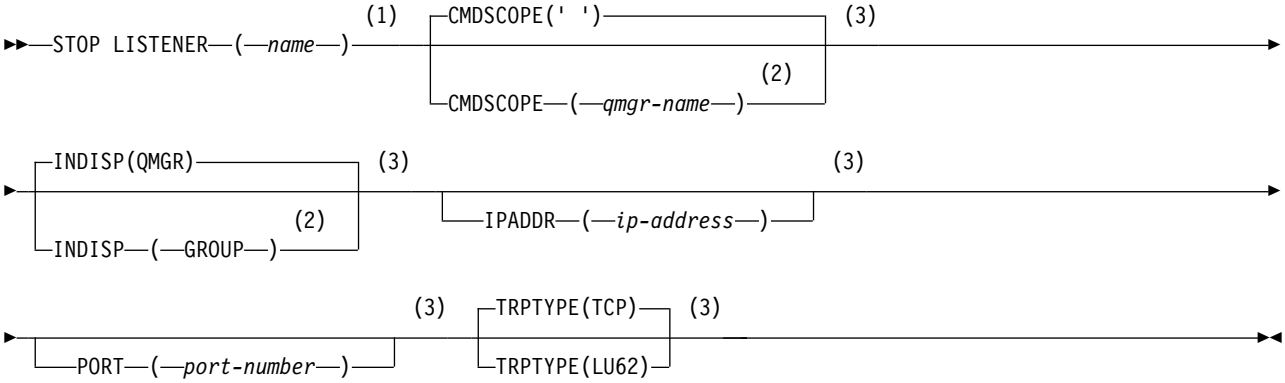
STOP LISTENER:

Use the MQSC command STOP LISTENER to stop a channel listener.

UNIX and Linux	Windows
✓	✓

Synonym: STOP LSTR

STOP LISTENER



Notes:

1 Not valid on z/OS.

2 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

3 Valid only on z/OS.

Usage notes

On z/OS:

- The command server and the channel initiator must be running.
- If a listener is listening on multiple addresses or ports, only the address and port combinations with the address, or port, specified are stopped.
- If a listener is listening on all addresses for a particular port, a stop request for a specific IPADDR with the same port fails.
- If neither an address nor a port is specified, all addresses and ports are stopped and the listener task ends.

Parameter descriptions for STOP LISTENER

(*name***)** Name of the listener to be stopped. If you specify this parameter, you cannot specify any other parameters.

This parameter is required on all platforms other than z/OS where it is not a supported parameter.

CMDSCOPE

This parameter specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name

The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

This parameter is valid only on z/OS.

INDISP
Specifies the disposition of the inbound transmissions that the listener handles. The possible values are:

QMGR
Handling for transmissions directed to the queue manager. This is the default.

GROUP
Handling for transmissions directed to the queue-sharing group. This is allowed only if there is a shared queue manager environment.

This parameter is valid only on z/OS.

IPADDR
IP address for TCP/IP specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric form. This is valid only if the transmission protocol (TRPTYPE) is TCP/IP.

This parameter is valid only on z/OS.

PORT
The port number for TCP/IP. This is the port number on which the listener is to stop listening. This is valid only if the transmission protocol is TCP/IP.

This parameter is valid only on z/OS.

TRPTYPE
Transmission protocol used. This is optional.

TCP TCP. This is the default if TRPTYPE is not specified.

LU62 SNA LU 6.2.

This parameter is valid only on z/OS.

The listener stops in quiesce mode (it disregards any further requests).

STOP SERVICE:

Use the MQSC command STOP SERVICE to stop a service.

UNIX and Linux	Windows
✓	✓

- Syntax diagram
- "Usage notes"
- "Parameter descriptions for STOP SERVICE" on page 794

Synonym:

STOP SERVICE

Usage notes

If the service is running, it is requested to stop. This command is processed asynchronously so might return before the service has stopped.

If the service that is requested to stop has no STOP command defined, an error is returned.

Parameter descriptions for STOP SERVICE

(*service-name***)**
The name of the service definition to be stopped. This is required. The name must that of an existing service on this queue manager.

Related reference:

"ALTER SERVICE" on page 416
Use the MQSC command ALTER SERVICE to alter the parameters of an existing WebSphere MQ service definition.

"START SERVICE" on page 785
Use the MQSC command START SERVICE to start a service. The identified service definition is started within the queue manager and inherits the environment and security variables of the queue manager.

Related information:

Working with services

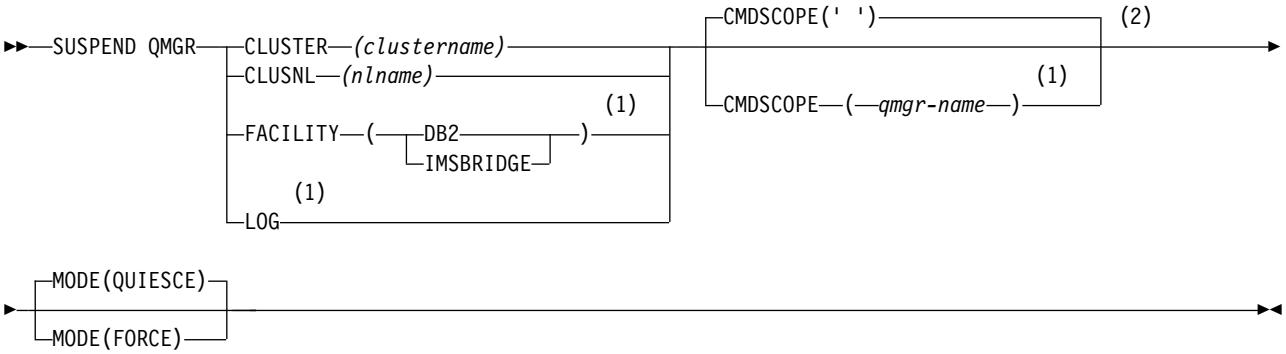
SUSPEND QMGR:

Use the MQSC command SUSPEND QMGR to advise other queue managers in a cluster to avoid sending messages to the local queue manager if possible, or to suspend logging and update activity for the queue manager until a subsequent RESUME QMGR command is issued. Its action can be reversed by the RESUME QMGR command. This command does not mean that the queue manager is disabled.

UNIX and Linux	Windows
✓	✓

Synonym: None

SUSPEND QMGR



Notes:

1 Valid only on z/OS.

2 Valid only on WebSphere MQ for z/OS when the queue manager is a member of a queue-sharing group.

Usage notes

On z/OS:

- If you define CLUSTER or CLUSNL, be aware of the following behavior:
  - The command fails if the channel initiator has not been started.
  - Any errors are reported to the system console where the channel initiator is running; they are not reported to the system that issued the command.
- The SUSPEND QMGR and RESUME QMGR commands are supported through the console only. However, all the other SUSPEND and RESUME commands are supported through the console and command server.

Parameter descriptions for SUSPEND QMGR

The SUSPEND QMGR with the CLUSTER or CLUSNL parameters to specify the cluster or clusters for which availability is suspended, how the suspension takes effect and, on z/OS, controls logging and update activity and how the command is executed when the queue manager is a member of a queue-sharing group.

You can use the SUSPEND QMGR FACILITY(DB2) to terminate the queue manager connection to Db2. This command might be useful if you want to apply service to Db2. Be aware, if you use this option then there is no access to Db2 resources, for example, large messages which might be offloaded to Db2 from a coupling facility.

You can use the SUSPEND QMGR FACILITY(IMSBRIDGE) to stop sending messages from the WebSphere MQ IMS bridge to IMS OTMA.

CLUSTER*(clustername)*
The name of the cluster for which availability is to be suspended.

CLUSNL*(nlname)*
The name of the namelist that specifies a list of clusters for which availability is to be suspended.

LOG Suspends logging and update activity for the queue manager until a subsequent RESUME request is issued. Any unwritten log buffers are externalized, a system checkpoint is taken (non-data sharing environment only), and the BSDS is updated with the high-written RBA before the update activity is suspended. A highlighted message (CSQJ372I) is issued and remains on the system console until update activity has been resumed. Valid on z/OS only. If LOG is specified, the command can be issued only from the z/OS system console.

This option is not permitted when a system quiesce is active by either the ARCHIVE LOG or STOP QMGR command.

Update activity remains suspended until a RESUME QMGR LOG or STOP QMGR command is issued.

This command must not be used during periods of high activity, or for long periods of time. Suspending update activity can cause timing-related events such as lock timeouts or WebSphere MQ diagnostic memory dumps when delays are detected.

CMDSCOPE
This parameter applies to z/OS only and specifies how the command is executed when the queue manager is a member of a queue-sharing group.

' ' The command is executed on the queue manager on which it was entered. This is the default value.

qmgr-name
The command is executed on the queue manager you specify, providing the queue manager is active within the queue-sharing group.

You can specify a queue manager name, other than the queue manager on which the command was entered, only if you are using a queue-sharing group environment and if the command server is enabled.

MODE

Specifies how the suspension of availability is to take effect:

QUIESCE

Other queue managers in the cluster are advised to avoid sending messages to the local queue manager if possible. It does not mean that the queue manager is disabled.

FORCE

All inbound channels from other queue managers in the cluster are stopped forcibly. This occurs only if the queue manager has also been forcibly suspended from all other clusters to which the channel belongs.

The MODE keyword is permitted only with CLUSTER or CLUSNL. It is not permitted with the LOG or FACILITY parameter.

Programmable command formats reference

Programmable Command Formats (PCFs) define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. PCFs simplify queue manager administration and other network administration.

For an introduction to PCFs, see Introduction to Programmable Command Formats.

For the full list of PCFs, see "Definitions of the Programmable Command Formats."

PCF commands and responses have a consistent structure including of a header and any number of parameter structures of defined types. For information about these structures, see "Structures for commands and responses" on page 1214.

For an example PCF, see "PCF example" on page 1241.

Related concepts:

"WebSphere MQ Control commands" on page 131
Find out how to use the WebSphere MQ control commands.

"MQSC reference" on page 276
Use MQSC commands to manage queue manager objects, including the queue manager itself, queues, process definitions, channels, client connection channels, listeners, services, namelists, clusters, and authentication information objects.

Definitions of the Programmable Command Formats

All the available Programmable Command Formats (PCFs) are listed including their parameters (required and optional), response data and error codes.

Following is the reference information for the Programmable Command Formats (PCFs) of commands and responses sent between a WebSphere MQ systems management application program and a WebSphere MQ queue manager.

How the definitions are shown:

The definitions of the Programmable Command Formats (PCFs) including their commands, responses, parameters, constants, and error codes are shown in a consistent format.

For each PCF command or response, there is a description of what the command or response does, giving the command identifier in parentheses. See Constants for all values of the command identifier. Each command description starts with a table that identifies the platforms on which the command is valid. For additional, more detailed, usage notes for each command, see the corresponding command description in the MQSC reference.

WebSphere MQ products, other than WebSphere MQ for z/OS, can use the WebSphere MQ Administration Interface (MQAI), which provides a simplified way for applications written in the C and Visual Basic programming language to build and send PCF commands. For information about the MQAI see the second section of this topic.

Commands

The *required parameters* and the *optional parameters* are listed. On platforms other than z/OS, the parameters ***must*** occur in the order:

1. All required parameters, in the order stated, followed by
2. Optional parameters as required, in any order, unless noted in the PCF definition.

On z/OS, the parameters can be in any order.

Responses

The response data attribute is *always returned* whether it is requested or not. This parameter is required to identify, uniquely, the object when there is a possibility of multiple reply messages being returned.

The other attributes shown are *returned if requested* as optional parameters on the command. The response data attributes are not returned in a defined order.

Parameters and response data

Each parameter name is followed by its structure name in parentheses (details are given in "Structures for commands and responses" on page 1214). The parameter identifier is given at the beginning of the description.

Constants

For the values of constants used by PCF commands and responses see Constants.

Informational messages

On z/OS, a number of command responses return a structure, MQIACF_COMMAND_INFO, with values that provide information about the command.

Table 93. MQIACF_COMMAND_INFO values

MQIACF_COMMAND_INFO value	Meaning
MQCMDI_CMDSCOPE_ACCEPTED	A command that specified *CommandScope* was entered. It has been passed to the one or more requested queue managers for processing
MQCMDI_CMDSCOPE_GENERATED	A command that specified *CommandScope* was generated in response to the command originally entered
MQCMDI_CMDSCOPE_COMPLETED	Processing for the command that specified *CommandScope* - either entered or generated by another command - has completed successfully on all requested queue managers
MQCMDI_QSG_DISP_COMPLETED	Processing for the command that refers to an object with the indicated disposition has completed successfully
MQCMDI_COMMAND_ACCEPTED	Initial processing for the command has completed successfully. The command requires further action by the channel initiator, for which a request has been queued. Messages reporting the success or otherwise of the action are be sent to the command issuer later
MQCMDI_CLUSTER_REQUEST_QUEUED	Initial processing for the command has completed successfully. The command requires further action by the cluster repository manager, for which a request has been queued
MQCMDI_CHANNEL_INIT_STARTED	A Start Channel Initiator command has been issued and the channel initiator address space has been started successfully
MQCMDI_RECOVER_STARTED	The queue manager has successfully started a task to process the Recover CF Structure command for the named structure
MQCMDI_BACKUP_STARTED	The queue manager has successfully started a task to process the Backup CF Structure command for the named structure

Table 93. MQIACF_COMMAND_INFO values (continued)

MQIACF_COMMAND_INFO value	Meaning
MQCMDI_RECOVER_COMPLETED	The named CF structure has been recovered successfully. The structure is available for use again
MQCMDI_SEC_TIMER_ZERO	The Change Security command was entered with the *SecurityInterval* attribute set to 0. This means that no user timeouts occur
MQCMDI_REFRESH_CONFIGURATION	A Change Queue Manager command has been issued that enables configuration events. Event messages need to be generated to ensure that the configuration information is complete and up to date
MQCMDI_IMS_BRIDGE_SUSPENDED	The MQ-IMS Bridge facility is suspended.
MQCMDI_DB2_SUSPENDED	The connection to DB2 is suspended
MQCMDI_DB2_OBSOLETE_MSGS	Obsolete DB2 messages exist in the queue-sharing group

Error codes

At the end of most command format definitions, there is a list of error codes that might be returned by that command.

Error codes applicable to all commands

In addition to those error codes listed under each command format, any command might return the following error codes in the response format header (descriptions of the MQRC_* error codes are given in the Reason codes:

Reason **(MQLONG)**
The value can be:

MQRC_NONE
(0, X'000') No reason to report.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_UNKNOWN_OBJECT_NAME
(2067, X'813') Attribute selector not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFBF_FILTER_VAL_LEN_ERROR
Filter value length not valid.

MQRCCF_CFBF_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFBF_OPERATOR_ERROR
Operator error.

MQRCCF_CFBF_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFBS_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFBS_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFBS_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFBS_STRING_LENGTH_ERROR
String length not valid.

MQRCCF_CFGR_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFGR_PARM_COUNT_ERROR
Parameter count not valid.

MQRCCF_CFGR_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFH_COMMAND_ERROR
Command identifier not valid.

MQRCCF_CFH_CONTROL_ERROR
Control option not valid.

MQRCCF_CFH_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFH_MSG_SEQ_NUMBER_ERR
Message sequence number not valid.

MQRCCF_CFH_PARM_COUNT_ERROR
Parameter count not valid.

MQRCCF_CFH_TYPE_ERROR
Type not valid.

MQRCCF_CFH_VERSION_ERROR
Structure version number is not valid.

MQRCCF_CFIF_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIF_OPERATOR_ERROR
Operator error.

MQRCCF_CFIF_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFSF_FILTER_VAL_LEN_ERROR
Filter value length not valid.

MQRCCF_CFSF_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFSF_OPERATOR_ERROR
Operator error.

MQRCCF_CFSF_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFSL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFSL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFSL_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFSL_STRING_LENGTH_ERROR
String length value not valid.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

MQRCCF_CFST_CONFLICTING_PARM
Conflicting parameters.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFST_STRING_LENGTH_ERROR
String length value not valid.

MQRCCF_COMMAND_FAILED
Command failed.

MQRCCF_ENCODING_ERROR
Encoding error.

MQRCCF_MD_FORMAT_ERROR
Format not valid.

MQRCCF_MSG_SEQ_NUMBER_ERROR
Message sequence number not valid.

MQRCCF_MSG_TRUNCATED
Message truncated.

MQRCCF_MSG_LENGTH_ERROR
Message length not valid.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too large.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PARM_SYNTAX_ERROR
Syntax error found in parameter.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

PCF commands and responses in groups:

In this product documentation, the commands and data responses are given in alphabetical order.

They can be usefully grouped as follows:

Authentication Information commands
- "Change, Copy, and Create Authentication Information Object" on page 810
- "Delete Authentication Information Object" on page 925
- "Inquire Authentication Information Object" on page 941
- "Inquire Authentication Information Object Names" on page 944

Authority Record commands
- "Delete Authority Record" on page 926
- "Inquire Authority Records" on page 947
- "Inquire Authority Service" on page 953
- "Inquire Entity Authority" on page 1043
- "Set Authority Record" on page 1188

CF commands
- Backup CF Structure
-
-
-
-
-
-

Channel commands

- "Change, Copy, and Create Channel" on page 813
- "Delete Channel" on page 928
- "Inquire Channel" on page 955
-
- "Inquire Channel Names" on page 989
- "Inquire Channel Status" on page 991
- "Ping Channel" on page 1168
- "Reset Channel" on page 1179
- "Resolve Channel" on page 1185
- "Start Channel" on page 1198
- "Start Channel Initiator" on page 1202
- "Stop Channel" on page 1205
-

Channel commands (MQTT)

- "Change, Copy, and Create Channel (MQTT)" on page 846
- "Delete Channel (MQTT)" on page 930
- "Inquire Channel (MQTT)" on page 962
- "Inquire Channel Status (MQTT)" on page 1002
- "Purge Channel" on page 1172
- "Start Channel (MQTT)" on page 1201
- "Stop Channel (MQTT)" on page 1209

Channel Authentication commands

- "Inquire Channel Authentication Records" on page 975
- "Set Channel Authentication Record" on page 1193

Channel Listener commands

- "Change, Copy, and Create Channel Listener" on page 851
- "Delete Channel Listener" on page 931
- "Inquire Channel Listener" on page 980
- "Inquire Channel Listener Status" on page 984
- "Start Channel Listener" on page 1203
- "Stop Channel Listener" on page 1210

Cluster commands

- "Inquire Cluster Queue Manager" on page 1016
- "Refresh Cluster" on page 1173
- "Reset Cluster" on page 1181
- "Resume Queue Manager Cluster" on page 1187
- "Suspend Queue Manager Cluster" on page 1213

Communication Information commands

- "Change, Copy, and Create Communication Information Object" on page 854
- "Delete Communication Information Object" on page 932
- "Inquire Communication Information Object" on page 1028

Connection commands

- "Inquire Connection" on page 1032
- "Stop Connection" on page 1212

Escape command

- "Escape" on page 939

Namelist commands

- "Change, Copy, and Create Namelist" on page 857
- "Delete Namelist" on page 932
- "Inquire Namelist" on page 1048
- "Inquire Namelist Names" on page 1052

Process commands

- "Change, Copy, and Create Process" on page 860
- "Delete Process" on page 933
- "Inquire Process" on page 1054
- "Inquire Process Names" on page 1058

Publish/subscribe commands

- "Change, Copy, and Create Subscription" on page 911
- "Change, Copy, and Create Topic" on page 915
- "Clear Topic String" on page 924
- "Delete Subscription" on page 937
- "Delete Topic" on page 938
- "Inquire Pub/Sub Status" on page 1060
- "Inquire Subscription" on page 1140
- "Inquire Subscription Status" on page 1147
- "Inquire Topic" on page 1150
- "Inquire Topic Names" on page 1159
- "Inquire Topic Status" on page 1161

Queue commands

- "Change, Copy, and Create Queue" on page 865
- "Clear Queue" on page 923
- "Delete Queue" on page 934
- "Inquire Queue" on page 1064
- "Inquire Queue Names" on page 1119
- "Inquire Queue Status" on page 1121
-
- "Reset Queue Statistics" on page 1183

Queue Manager commands

- "Change Queue Manager" on page 882
- "Inquire Queue Manager" on page 1083
- "Inquire Queue Manager Status" on page 1115
- "Ping Queue Manager" on page 1172

- "Refresh Queue Manager" on page 1174
- "Reset Queue Manager" on page 1182
-
-

Security commands
-
-
- "Refresh Security" on page 1177
-

Service commands
- "Change, Copy, and Create Service" on page 908
- "Delete Service" on page 937
- "Inquire Service" on page 1133
- "Inquire Service Status" on page 1136
- "Start Service" on page 1205
- "Stop Service" on page 1212

SMDS commands
-
-
-
-
-
-

Storage class commands
-
-
-
-

System commands
-
-
- Inquire Group
-
-
-
-

Data responses to commands
- "Escape (Response)" on page 940
-
- "Inquire Authentication Information Object (Response)" on page 943

- “Inquire Authentication Information Object Names (Response)” on page 946
- “Inquire Authority Records (Response)” on page 950
- “Inquire Authority Service (Response)” on page 954
-
-
-
- “Inquire Channel (Response)” on page 964
- “Inquire Channel Authentication Records (Response)” on page 978
-
- “Inquire Channel Listener (Response)” on page 982
- “Inquire Channel Listener Status (Response)” on page 986
- “Inquire Channel Names (Response)” on page 991
- “Inquire Channel Status (Response)” on page 1004
- “Inquire Channel Status (Response)” on page 1014
- “Inquire Cluster Queue Manager (Response)” on page 1021
- “Inquire Communication Information Object (Response)” on page 1030
- “Inquire Connection (Response)” on page 1036
- “Inquire Entity Authority (Response)” on page 1046
-
-
- “Inquire Namelist (Response)” on page 1051
- “Inquire Namelist Names (Response)” on page 1053
- “Inquire Process (Response)” on page 1056
- “Inquire Process Names (Response)” on page 1059
- “Inquire Pub/Sub Status (Response)” on page 1061
- “Inquire Queue (Response)” on page 1072
- “Inquire Queue Manager (Response)” on page 1093
- “Inquire Queue Manager Status (Response)” on page 1117
- “Inquire Queue Names (Response)” on page 1120
- “Reset Queue Statistics (Response)” on page 1184
- “Inquire Queue Status (Response)” on page 1126
-
- “Inquire Service (Response)” on page 1134
- “Inquire Service Status (Response)” on page 1138
-
-
-
-
- “Inquire Subscription (Response)” on page 1143
- “Inquire Subscription Status (Response)” on page 1149
-
- “Inquire Topic (Response)” on page 1154
- “Inquire Topic Names (Response)” on page 1160
- “Inquire Topic Status (Response)” on page 1162
-

Change, Copy, and Create Authentication Information Object:

The Change authentication information command changes attributes of an existing authentication information object. The Create and Copy authentication information commands create new authentication information objects - the Copy command uses attribute values of an existing object.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

The Change authentication information (MQCMD_CHANGE_AUTH_INFO) command changes the specified attributes in an authentication information object. For any optional parameters that are omitted, the value does not change.

The Copy authentication information (MQCMD_COPY_AUTH_INFO) command creates new authentication information object using, for attributes not specified in the command, the attribute values of an existing authentication information object.

The Create authentication information (MQCMD_CREATE_AUTH_INFO) command creates an authentication information object. Any attributes that are not defined explicitly are set to the default values on the destination queue manager. A system default authentication information object exists and default values are taken from it.

Required parameters (Change authentication information)

AuthInfoName **(MQCFST)**
The authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value can be:

MQAIT_CRL_LDAP
This defines this authentication information object as specifying an LDAP server containing Certificate Revocation Lists.

MQAIT_OCSP
This value defines this authentication information object as specifying certificate revocation checking using OCSP.

AuthInfoType MQAIT_OCSP does not apply for use on IBM i or z/OS queue managers, but it can be specified on those platforms to be copied to the client channel definition table for client use.

See Security for more information.

Required parameters (Copy authentication information)

FromAuthInfoName (MQCFST)
The name of the authentication information object definition to be copied from (parameter identifier: MQCACF_FROM_AUTH_INFO_NAME).

On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToAuthInfoName* and the disposition of MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

ToAuthInfoName (MQCFST)
The name of the authentication information object to copy to (parameter identifier: MQCACF_TO_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE). The value must match the AuthInfoType of the authentication information object from which you are copying.

The value can be:

MQAIT_CRL_LDAP
This value defines this authentication information object as specifying Certificate Revocation Lists that are held on LDAP.

MQAIT_OCSP
This value defines this authentication information object as specifying certificate revocation checking using OCSP.

See Security for more information.

Required parameters (Create authentication information)

AuthInfoName **(MQCFST)**
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType **(MQCFIN)**
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

The following values are accepted:

MQAIT_CRL_LDAP
This value defines this authentication information object as specifying an LDAP server containing Certificate Revocation Lists.

MQAIT_OCSP
This value defines this authentication information object as specifying certificate revocation checking using OCSP.

An authentication information object with AuthInfoType MQAIT_OCSP does not apply for use on IBM i or z/OS queue managers, but it can be specified on those platforms to be copied to the client channel definition table for client use.

See Security for more information.

Optional parameters (Change, Copy, and Create Authentication Information Object)

AuthInfoConnName (MQCFST)
The connection name of the authentication information object (parameter identifier: MQCA_AUTH_INFO_CONN_NAME).

On platforms other than z/OS, the maximum length is MQ_AUTH_INFO_CONN_NAME_LENGTH. On z/OS, it is MQ_LOCAL_ADDRESS_LENGTH.

This parameter is relevant only when AuthInfoType is set to MQAIT_CRL_LDAP, when it is required.

AuthInfoDesc (MQCFST)
The description of the authentication information object(parameter identifier: MQCA_AUTH_INFO_DESC).

The maximum length is MQ_AUTH_INFO_DESC_LENGTH.

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

LDAPPassword (MQCFST)
The LDAP password (parameter identifier: MQCA_LDAP_PASSWORD).

The maximum length is MQ_LDAP_PASSWORD_LENGTH.

This parameter is relevant only when AuthInfoType is set to MQAIT_CRL_LDAP.

LDAPUserName (MQCFST)
The LDAP user name (parameter identifier: MQCA_LDAP_USER_NAME).

On platforms other than z/OS, the maximum length is MQ_DISTINGUISHED_NAME_LENGTH. On z/OS, it is MQ_SHORT_DNAME_LENGTH.

This parameter is relevant only when AuthInfoType is set to MQAIT_CRL_LDAP.

OCSPResponderURL (MQCFST)
The URL at which the OCSP responder can be contacted (parameter identifier: MQCA_AUTH_INFO_OCSP_URL).

This parameter is relevant only when AuthInfoType is set to MQAIT_OCSP, when it is required.

This field is case-sensitive. It must start with the string http:// in lowercase. The rest of the URL might be case sensitive, depending on the OCSP server implementation.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OSonly.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameter MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToAuthInfoName* object (for Copy) or the *AuthInfoName* object (for Create).

QSGDisposition	Change	Copy, Create

MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.
	
	If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they refresh local copies on page set zero:
	`DEFINE AUTHINFO(name)`
	`REPLACE QSGDISP(COPY)`
	
	The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
		The object definition resides in the shared repository. This definition is allowed only if the queue manager is in a queue-sharing group.
		
		If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they make or refresh local copies on page set zero:
		`DEFINE AUTHINFO(name)`
		`REPLACE QSGDISP(COPY)`
		
		The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR, or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

Replace **(MQCFIN)**
Replace attributes (parameter identifier: MQIACF_REPLACE).

If an Authentication Information object with the same name as AuthInfoName or ToAuthInfoName exists, it specifies whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition

MQRP_NO
Do not replace existing definition

Change, Copy, and Create Channel:

The Change Channel command changes existing channel definitions. The Copy and Create Channel commands create new channel definitions - the Copy command uses attribute values of an existing channel definition.

HP Integrity NonStop Server	IBM i	UNIX and Linux	Windows	z/OS
✓	✓	✓	✓	✓

The Change Channel (MQCMD_CHANGE_CHANNEL) command changes the specified attributes in a channel definition. For any optional parameters that are omitted, the value does not change.

The Copy Channel (MQCMD_COPY_CHANNEL) command creates new channel definition using, for attributes not specified in the command, the attribute values of an existing channel definition.

The Create Channel (MQCMD_CREATE_CHANNEL) command creates a WebSphere MQ channel definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager. If a system default channel exists for the type of channel being created, the default values are taken from there.

Table 94 shows the parameters that are applicable to each type of channel.

Table 94. Change, Copy, Create Channel parameters

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
BatchHeartBeat	✓	✓					✓	✓
BatchInterval	✓	✓					✓	✓
BatchDataLimit	✓	✓					✓	✓
BatchSize	✓	✓	✓	✓			✓	✓
ChannelDesc	✓	✓	✓	✓	✓	✓	✓	✓
ChannelMonitoring	✓	✓	✓	✓		✓	✓	✓
ChannelStatistics	✓	✓	✓	✓			✓	✓
ChannelName[1]	✓	✓	✓	✓	✓	✓	✓	✓
ChannelType[3]	✓	✓	✓	✓	✓	✓	✓	✓
ClientChannelWeight					✓			
ClusterName							✓	✓
ClusterNameList							✓	✓
CLWLChannelPriority							✓	✓
CLWLChannelRank							✓	✓
CLWLChannelWeight							✓	✓
CommandScope	✓	✓	✓	✓	✓	✓	✓	✓
ConnectionAffinity					✓			
ConnectionName	✓	✓		✓	✓		✓	✓

Table 94. Change, Copy, Create Channel parameters (continued)

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
DataConversion	✓	✓		✓	✓		✓	✓
DefaultChannelDisposition	✓	✓	✓	✓		✓	✓	✓
DefReconnect					✓			
DiscInterval	✓	✓				✓	✓	✓
FromChannelName[2]	✓	✓	✓	✓	✓	✓	✓	✓
HeaderCompession	✓	✓	✓	✓	✓	✓	✓	✓
HeartBeatInterval	✓	✓	✓	✓	✓	✓	✓	✓
KeepAliveInterval	✓	✓	✓	✓	✓	✓	✓	✓
LocalAddress	✓	✓		✓	✓		✓	✓
LongRetryCount	✓	✓					✓	✓
LongRetryInterval	✓	✓					✓	✓
MaxInstances						✓		
MaxInstancesPerClient						✓		
MaxMsgLength	✓	✓	✓	✓	✓	✓	✓	✓
MCAName	✓	✓		✓			✓	
MCAType	✓	✓		✓			✓	✓
MCAUserIdentifier			✓	✓		✓		✓
MessageCompression	✓	✓	✓	✓	✓	✓	✓	✓
ModeName	✓	✓		✓	✓		✓	✓
MsgExit	✓	✓	✓	✓			✓	✓
MsgRetryCount			✓	✓				✓
MsgRetryExit			✓	✓				✓
MsgRetryInterval			✓	✓				✓
MsgRetryUserData			✓	✓				✓

Table 94. Change, Copy, Create Channel parameters (continued)

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
MsgUserData	✓	✓	✓	✓			✓	✓
NetworkPriority								✓
NonPersistentMsgSpeed	✓	✓	✓	✓			✓	✓
Password	✓	✓		✓	✓		✓	
PropertyControl	✓	✓					✓	✓
PutAuthority			✓	✓		✓		✓
QMgrName					✓			
QSGDisposition	✓	✓	✓	✓	✓	✓	✓	✓
ReceiveExit	✓	✓	✓	✓	✓	✓	✓	✓
ReceiveUserData	✓	✓	✓	✓	✓	✓	✓	✓
Replace	✓	✓	✓	✓	✓	✓	✓	✓
SecurityExit	✓	✓	✓	✓	✓	✓	✓	✓
SecurityUserData	✓	✓	✓	✓	✓	✓	✓	✓
SendExit	✓	✓	✓	✓	✓	✓	✓	✓
SendUserData	✓	✓	✓	✓	✓	✓	✓	✓
SeqNumberWrap	✓	✓	✓	✓			✓	✓
SharingConversations					✓	✓		
ShortRetryCount	✓	✓					✓	✓
ShortRetryInterval	✓	✓					✓	✓
SSLCipherSpec	✓	✓	✓	✓	✓	✓	✓	✓
SSLClientAuth		✓	✓	✓		✓		✓
SSLPeerName	✓	✓	✓	✓	✓	✓	✓	✓
ToChannelName[2]	✓	✓	✓	✓	✓	✓	✓	✓
TpName	✓	✓		✓	✓	✓	✓	✓

Table 94. Change, Copy, Create Channel parameters (continued)

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
TransportType	✓	✓	✓	✓	✓	✓	✓	✓
UseDLQ	✓	✓	✓	✓			✓	✓
UserIdentifier	✓	✓		✓	✓		✓	
XmitQName	✓	✓						

Note:

1. Required parameter on Change and Create Channel commands.
2. Required parameter on Copy Channel command.
3. Required parameter on Change, Create, and Copy Channel commands.
4. PUTAUT is valid for a channel type of SVRCONN on z/OS only.
5. Required parameter on Create Channel command if `TrpType` is TCP.
6. Required parameter on Create Channel command for a channel type of MQTT.

Required parameters (Change, Create Channel)

ChannelName **(MQCFST)**

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies the name of the channel definition to be changed, or created

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required on all types of channel; on a CLUSSDR it can be different from on the other channel types. If your convention for naming channels includes the name of the queue manager, you can make a CLUSSDR definition using the `+QMNAME+` construction, and WebSphere MQ substitutes the correct repository queue manager name in place of `+QMNAME+` . This facility applies to AIX , HP-UX, Linux , IBM i , Solaris, and Windows only. See Configuring a queue manager cluster for more details.

ChannelType **(MQCFIN)**

Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed, copied, or created. The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

Required parameters (Copy Channel)

FromChannelName **(MQCFST)**
From channel name (parameter identifier: MQCACF_FROM_CHANNEL_NAME).

The name of the existing channel definition that contains values for the attributes that are not specified in this command.

On z/OS , the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition* . In this case, an object with the name specified by *ToChannelName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed, copied, or created. The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

ToChannelName **(MQCFST)**
To channel name (parameter identifier: MQCACF_TO_CHANNEL_NAME).

The name of the new channel definition.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Channel names must be unique; if a channel definition with this name exists, the value of *Replace* must be MQRP_YES. The channel type of the existing channel definition must be the same as the channel type of the new channel definition otherwise it cannot be replaced.

Optional parameters (Change, Copy, and Create Channel)

BatchHeartbeat **(MQCFIN)**
The batch heartbeat interval (parameter identifier: MQIACH_BATCH_HB).

Batch heartbeating allows sender-type channels to determine whether the remote channel instance is still active, before going in-doubt. The value can be in the range 0 - 999999. A value of 0 indicates that batch heart-eating is not to be used. Batch heartbeat is measured in milliseconds.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

BatchInterval **(MQCFIN)**
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

This interval is the approximate time in milliseconds that a channel keeps a batch open, if fewer than *BatchSize* messages have been transmitted in the current batch.

If *BatchInterval* is greater than zero, the batch is terminated by whichever of the following situations occurs first:

- *BatchSize* messages have been sent, or
- *BatchInterval* milliseconds have elapsed since the start of the batch.

If *BatchInterval* is zero, the batch is terminated by whichever of the following situations occurs first:

- *BatchSize* messages have been sent, or
- *BatchDataLimit* bytes have been sent, or
- the transmission queue becomes empty.

BatchInterval must be in the range 0 - 999999999.

This parameter applies only to channels with a *ChannelType* of: MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

BatchDataLimit **(MQCFIN)**

Batch data limit (parameter identifier: MQIACH_BATCH_DATA_LIMIT).

The limit, in kilobytes, of the amount of data that can be sent through a channel before taking a sync point. A sync point is taken after the message that caused the limit to be reached has flowed across the channel. A value of zero in this attribute means that no data limit is applied to batches over this channel.

The value must be in the range 0 - 999999. The default value is 5000.

This parameter is supported on all platforms.

This parameter only applies to channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSRCVR, or MQCHT_CLUSSDR.

BatchSize **(MQCFIN)**
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

The maximum number of messages that must be sent through a channel before a checkpoint is taken.

The batch size which is used is the lowest of the following:

- The *BatchSize* of the sending channel
- The *BatchSize* of the receiving channel
- The maximum number of uncommitted messages at the sending queue manager
- The maximum number of uncommitted messages at the receiving queue manager

The maximum number of uncommitted messages is specified by the *MaxUncommittedMsgs* parameter of the Change Queue Manager command.

Specify a value in the range 1 - 9999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

ChannelDesc **(MQCFST)**
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

Use characters from the character set, identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing, to ensure that the text is translated correctly.

ChannelMonitoring **(MQCFIN)**
Online monitoring data collection (parameter identifier: MQIA_MONITORING_CHANNEL).

Specifies whether online monitoring data is to be collected and, if so, the rate at which the data is collected. The value can be:

MQMON_OFF
Online monitoring data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's *ChannelMonitoring* parameter is inherited by the channel.

MQMON_LOW
If the value of the queue manager's *ChannelMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MQMON_MEDIUM
If the value of the queue manager's *ChannelMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.

MQMON_HIGH
If the value of the queue manager's *ChannelMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

ChannelStatistics **(MQCFIN)**
Statistics data collection (parameter identifier: MQIA_STATISTICS_CHANNEL).

Specifies whether statistics data is to be collected and, if so, the rate at which the data is collected. The value can be:

MQMON_OFF
Statistics data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's *ChannelStatistics* parameter is inherited by the channel.

MQMON_LOW
If the value of the queue manager's *ChannelStatistics* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection, for this channel.

MQMON_MEDIUM
If the value of the queue manager's *ChannelStatistics* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection, for this channel.

MQMON_HIGH
If the value of the queue manager's *ChannelStatistics* parameter is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection, for this channel.

This parameter is valid only on AIX , HP-UX, Linux , IBM i , Solaris, and Windows .

ClientChannelWeight **(MQCFIN)**
Client Channel Weight (parameter identifier: MQIACH_CLIENT_CHANNEL_WEIGHT).

The client channel weighting attribute is used so client channel definitions can be selected at random, with the larger weightings having a higher probability of selection, when more than one suitable definition is available.

Specify a value in the range 0 - 99. The default is 0.

This parameter is only valid for channels with a ChannelType of MQCHT_CLNTCONN

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to which the channel belongs.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

Only one of the values of *ClusterName* and *ClusterNamelist* can be nonblank; the other must be blank.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist **(MQCFST)**
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name, of the namelist, that specifies a list of clusters to which the channel belongs.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

Only one of the values of *ClusterName* and *ClusterNamelist* can be nonblank; the other must be blank.

CLWLChannelPriority **(MQCFIN)**
Channel priority for the purposes of cluster workload distribution (parameter identifier: MQIACH_CLWL_CHANNEL_PRIORITY).

Specify a value in the range 0 - 9 where 0 is the lowest priority and 9 is the highest.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

CLWLChannelRank **(MQCFIN)**
Channel rank for the purposes of cluster workload distribution (parameter identifier: MQIACH_CLWL_CHANNEL_RANK).

Specify a value in the range 0 - 9 where 0 is the lowest priority and 9 is the highest.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

CLWLChannelWeight **(MQCFIN)**
Channel weighting for the purposes of cluster workload distribution (parameter identifier: MQIACH_CLWL_CHANNEL_WEIGHT).

Specify a weighting for the channel for use in workload management. Specify a value in the range 1 - 99 where 1 is the lowest priority and 99 is the highest.

This parameter applies only to channels with a *ChannelType* of:

- MQCHT_CLUSSDR
- MQCHT_CLUSRCVR

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ConnectionAffinity **(MQCFIN)**
Channel Affinity (parameter identifier: MQIACH_CONNECTION_AFFINITY)

The channel affinity attribute specifies whether client applications that connect multiple times using the same queue manager name, use the same client channel. The value can be:

MQCAFTY_PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any zero ClientChannelWeight definitions first in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful nonzero ClientChannelWeight definitions are moved to the end of the list. Zero ClientChannelWeight definitions remain at the start of the list and are selected first for each connection. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created. Each client process with the same host name creates the same list.

This value is the default value.

MQCAFTY_NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process independently select an applicable definition based on the weighting with any applicable zero ClientChannelWeight definitions selected first in alphabetical order. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created.

This parameter is only valid for channels with a ChannelType of MQCHT_CLNTCONN.

ConnectionName **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

On platforms other than z/OS , the maximum length of the string is 264. On z/OS , it is 48.

Specify *ConnectionName* as a comma-separated list of names of machines for the stated *TransportType*. Typically, only one machine name is required. You can provide multiple machine names to configure multiple connections with the same properties. The connections are tried in the order they are specified in the connection list until a connection is successfully established. If no connection is successful, the channel starts to try processing again. Connection lists are an alternative to queue manager groups to configure connections for reconnectable clients, and also to configure channel connections to multi-instance queue managers.

Specify the name of the machine as required for the stated *TransportType*:

- For MQXPT_LU62 on IBM i , and UNIX systems, specify the name of the CPI-C communications side object. On Windows specify the CPI-C symbolic destination name.

  On z/OS , there are two forms in which to specify the value:

  Logical unit name
  The logical unit information for the queue manager, comprising the logical unit name, TP name, and optional mode name. This name can be specified in one of three forms:

Form	Example
luname	IGY12355
luname/TPname	IGY12345/APING
luname/TPname/modename	IGY12345/APINGD/#INTER

  For the first form, the TP name and mode name must be specified for the *TpName* and *ModeName* parameters; otherwise these parameters must be blank.

  Note: For client-connection channels, only the first form is allowed.

  Symbolic name
  The symbolic destination name for the logical unit information for the queue manager, as defined in the side information data set. The *TpName* and *ModeName* parameters must be blank.

  Note: For cluster-receiver channels, the side information is on the other queue managers in the cluster. Alternatively, in this case it can be a name that a channel auto-definition exit can resolve into the appropriate logical unit information for the local queue manager.

  The specified or implied LU name can be that of a VTAM generic resources group.
- For MQXPT_TCP, you can specify a connection name, or a connection list, containing the host name or the network address of the remote machine. Separate connection names in a connection list with commas.

  On AIX, HP-UX, IBM i, Linux, Solaris, and Windows platforms, the TCP/IP connection name parameter of a cluster-receiver channel is optional. If you leave the connection name blank, WebSphere MQ generates a connection name for you, assuming the default port and using the current IP address of the system. You can override the default port number, but still use the current IP address of the system. For each connection name leave the IP name blank, and provide the port number in parentheses; for example:

  ```
  (1415)
  ```

  The generated CONNAME is always in the dotted decimal (IPv4) or hexadecimal (IPv6) form, rather than in the form of an alphanumeric DNS host name.
- For MQXPT_NETBIOS specify the NetBIOS station name.
- For MQXPT_SPX specify the 4 byte network address, the 6 byte node address, and the 2 byte socket number. These values must be entered in hexadecimal, with a period separating the network and node addresses. The socket number must be enclosed in brackets, for example:

  ```
  0a0b0c0d.804abcde23a1(5e86)
  ```

If the socket number is omitted, the WebSphere MQ default value (5e86 hex) is assumed.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

Note: If you are using clustering between IPv6 -only and IPv4 -only queue managers, do not specify an IPv6 network address as the *ConnectionName* for cluster-receiver channels. A queue manager that is capable only of IPv4 communication is unable to start a cluster sender channel definition that specifies the *ConnectionName* in IPv6 hexadecimal form. Consider, instead, using host names in a heterogeneous IP environment.

DataConversion **(MQCFIN)**
Whether sender must convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DefaultChannelDisposition **(MQCFIN)**
Intended disposition of the channel when activated or started (parameter identifier: MQIACH_DEF_CHANNEL_DISP).

This parameter applies to z/OS only.

The value can be:

MQCHLD_PRIVATE
The intended use of the object is as a private channel.

This value is the default value.

MQCHLD_FIXSHARED
The intended use of the object is as a fixshared channel.

MQCHLD_SHARED
The intended use of the object is as a shared channel.

DefReconnect **(MQCFIN)**
Client channel default reconnection option (parameter identifier: MQIACH_DEF_RECONNECT).

The default automatic client reconnection option. You can configure a IBM WebSphere MQ MQI client to automatically reconnect a client application. The IBM WebSphere MQ MQI client tries to reconnect to a queue manager after a connection failure. It tries to reconnect without the application client issuing an MQCONN or MQCONNX MQI call.

MQRCN_NO
MQRCN_NO is the default value.

Unless overridden by MQCONNX, the client is not reconnected automatically.

MQRCN_YES
Unless overridden by MQCONNX, the client reconnects automatically.

MQRCN_Q_MGR
Unless overridden by MQCONNX, the client reconnects automatically, but only to the same queue manager. The QMGR option has the same effect as MQCNO_RECONNECT_Q_MGR.

MQRCN_DISABLED
Reconnection is disabled, even if requested by the client program using the MQCONNX MQI call.

Table 95. Automatic reconnection depends on the values set in the application and in the channel definition

DefReconnect	Reconnection options set in the application			
	MQCNO_RECONNECT	MQCNO_RECONNECT_Q_MGR	MQCNO_RECONNECT_AS_DEF	MQCNO_RECONNECT_DISABLED
MQRCN_NO	YES	QMGR	NO	NO
MQRCN_YES	YES	QMGR	YES	NO
MQRCN_Q_MGR	YES	QMGR	QMGR	NO
MQRCN_DISABLED	NO	NO	NO	NO

This parameter is valid only for a *ChannelType* value of MQCHT_CLNTCONN.

DiscInterval **(MQCFIN)**
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

This interval defines the maximum number of seconds that the channel waits for messages to be put on a transmission queue before terminating the channel. A value of zero causes the message channel agent to wait indefinitely.

Specify a value in the range 0 - 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER MQCHT_SERVER, MQCHT_SVRCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

For server-connection channels using the TCP protocol, this interval is the minimum time in seconds for which the server-connection channel instance remains active without any communication from its partner client. A value of zero disables this disconnect processing. The server-connection inactivity interval only applies between MQ API calls from a client, so no client is disconnected during an extended MQGET with wait call. This attribute is ignored for server-connection channels using protocols other than TCP.

HeaderCompression **(MQCFIL)**
Header data compression techniques supported by the channel (parameter identifier: MQIACH_HDR_COMPRESSION).

The list of header data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used.

The mutually supported compression techniques of the channel are passed to the message exit of the sending channel where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

Specify one or more of:

MQCOMPRESS_NONE
No header data compression is performed. This value is the default value.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval **(MQCFIN)**
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

The interpretation of this parameter depends on the channel type, as follows:

- For a channel type of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR, this interval is the time in

seconds between heartbeat flows passed from the sending MCA when there are no messages on the transmission queue. This interval gives the receiving MCA the opportunity to quiesce the channel. To be useful, *HeartbeatInterval* must be less than *DiscInterval*. However, the only check is that the value is within the permitted range.

This type of heartbeat is supported in the following environments: AIX , HP-UX, IBM i , Solaris, Windows , and z/OS .

- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN, this interval is the time in seconds between heartbeat flows passed from the server MCA when that MCA has issued an MQGET call with the MQGMO_WAIT option on behalf of a client application. This interval allows the server MCA to handle situations where the client connection fails during an MQGET with MQGMO_WAIT.

  This type of heartbeat is supported in the following environments: AIX , HP-UX, IBM i , Solaris, Windows , Linux, and z/OS .

The value must be in the range 0 - 999 999. A value of 0 means that no heartbeat exchange occurs. The value that is used is the larger of the values specified at the sending side and receiving side.

KeepAliveInterval **(MQCFIN)**
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

Specifies the value passed to the communications stack for KeepAlive timing for the channel.

For this attribute to be effective, TCP/IP keepalive must be enabled. On z/OS , you enable TCP/IP keepalive by issuing the Change Queue Manager command with a value of MQTCPKEEP in the *TCPKeepAlive* parameter; if the *TCPKeepAlive* queue manager parameter has a value of MQTCPKEEP_NO, the value is ignored, and the KeepAlive facility is not used. On other platforms, TCP/IP keepalive is enabled when the KEEPALIVE=YES parameter is specified in the TCP stanza in the distributed queuing configuration file, qm.ini, or through the WebSphere MQ Explorer. Keepalive must also be switched on within TCP/IP itself, using the TCP profile configuration data set.

Although this parameter is available on all platforms, its setting is implemented only on z/OS . On platforms other than z/OS , you can access and modify the parameter, but it is only stored and forwarded; there is no functional implementation of the parameter. This parameter is useful in a clustered environment where a value set in a cluster-receiver channel definition on Solaris, for example, flows to (and is implemented by) z/OS queue managers that are in, or join, the cluster.

Specify either:

integer
The KeepAlive interval to be used, in seconds, in the range 0 - 99 999. If you specify a value of 0, the value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

MQKAI_AUTO
The KeepAlive interval is calculated based upon the negotiated heartbeat value as follows:

- If the negotiated *HeartbeatInterval* is greater than zero, KeepAlive interval is set to that value plus 60 seconds.
- If the negotiated *HeartbeatInterval* is zero, the value used is that specified by the INTERVAL statement in the TCP profile configuration data set.

On platforms other than z/OS , if you need the functionality provided by the *KeepAliveInterval* parameter, use the *HeartBeatInterval* parameter.

LocalAddress **(MQCFST)**
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

The value that you specify depends on the transport type *(TransportType)* to be used:

TCP/IP

The value is the optional IP address and optional port or port range to be used for outbound TCP/IP communications. The format for this information is as follows:

`LOCLADDR([ip-addr][(low-port[,high-port])]`*`[,[ip-addr][(low-port[,high-port])]]`*`)`

where `ip-addr` is specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric form, and `low-port` and `high-port` are port numbers enclosed in parentheses. All are optional.

Specify *`[,[ip-addr][(low-port[,high-port])]]`* multiple times for each additional local address. Use multiple local addresses if you want to specify a specific subset of local network adapters. You can also use *`[,[ip-addr][(low-port[,high-port])]]`* to represent a particular local network address on different servers that are part of a multi-instance queue manager configuration.

All Others

The value is ignored; no error is diagnosed.

Use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. This parameter is useful when a machine is connected to multiple networks with different IP addresses.

Examples of use

Value	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98 (1000)	Channel binds to this address and port 1000 locally
9.20.4.98 (1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to a port in the range 1000 - 2000 locally

This parameter is valid for the following channel types:

- MQCHT_SENDER
- MQCHT_SERVER
- MQCHT_REQUESTER
- MQCHT_CLNTCONN
- MQCHT_CLUSRCVR
- MQCHT_CLUSSDR

Note:

- Do not confuse this parameter with *ConnectionName* . The *LocalAddress* parameter specifies the characteristics of the local communications; the *ConnectionName* parameter specifies how to reach a remote queue manager.

`LongRetryCount` **(MQCFIN)**

Long retry count (parameter identifier: MQIACH_LONG_RETRY).

When a sender or server channel is attempting to connect to the remote machine, and the count specified by *`ShortRetryCount`* has been exhausted, this count specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by *`LongRetryInterval`* .

If this count is also exhausted without success, an error is logged to the operator, and the channel is stopped. The channel must later be restarted with a command (it is not started automatically by the

channel initiator), and it then makes only one attempt to connect, as it is assumed that the problem has now been cleared by the administrator. The retry sequence is not carried out again until after the channel has successfully connected.

Specify a value in the range 0 - 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

LongRetryInterval **(MQCFIN)**
Long timer (parameter identifier: MQIACH_LONG_TIMER).

Specifies the long retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine, after the count specified by *ShortRetryCount* has been exhausted.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 - 999 999. Values exceeding this value are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

MaxInstances **(MQCFIN)**
Maximum number of simultaneous instances of a server-connection channel (parameter identifier: MQIACH_MAX_INSTANCES).

Specify a value in the range 0 - 999 999 999.

The default value is 999 999 999.

A value of zero indicates that no client connections are allowed on the channel.

If the value is reduced below the number of instances of the server-connection channel that are currently running, the running channels are not affected. This parameter applies even if the value is zero. However, if the value is reduced below the number of instances of the server-connection channel that are currently running, then new instances cannot be started until sufficient existing instances have ceased to run.

If you do not have the Client Attachment feature installed, the attribute can be set from zero to five only on the SYSTEM.ADMIN.SVRCONN channel. A value greater than five is interpreted as zero without the Client Attachment feature installed.

This parameter is valid only for channels with a *ChannelType* value of MQCHT_SVRCONN.

MaxInstancesPerClient **(MQCFIN)**
Maximum number of simultaneous instances of a server-connection channel that can be started from a single client (parameter identifier: MQIACH_MAX_INSTS_PER_CLIENT). In this context, connections that originate from the same remote network address are regarded as coming from the same client.

Specify a value in the range 0 - 999 999 999.

The default value is 999 999 999.

A value of zero indicates that no client connections are allowed on the channel.

If the value is reduced below the number of instances of the server-connection channel that are currently running from individual clients, the running channels are not affected. This parameter applies even if the value is zero. However, if the value is reduced below the number of instances of the server-connection channel that are currently running from individual clients, new instances from those clients cannot start until sufficient existing instances have ceased to run.

If you do not have the Client Attachment feature installed, the attribute can be set from zero to five only on the SYSTEM.ADMIN.SVRCONN channel. A value greater than five is interpreted as zero without the Client Attachment feature installed.

This parameter is valid only for channels with a *ChannelType* value of MQCHT_SVRCONN.

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

Specifies the maximum message length that can be transmitted on the channel. This value is compared with the value for the remote channel and the actual maximum is the lower of the two values.

The value zero means the maximum message length for the queue manager.

The lower limit for this parameter is 0. The maximum message length is 100 MB (104 857 600 bytes).

MCAName **(MQCFST)**
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL). For more details, see Channel authentication records

This parameter is reserved, and if specified can be set only to blanks.

The maximum length of the string is MQ_MCA_NAME_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

MCAType **(MQCFIN)**
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

Specifies the type of the message channel agent program.

On AIX , HP-UX, IBM i , Solaris, Windows, and Linux , this parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, or MQCHT_CLUSSDR.

On z/OS , this parameter is valid only for a *ChannelType* value of MQCHT_CLURCVR.

The value can be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread.

MCAUserIdentifier **(MQCFST)**
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

If this parameter is nonblank, it is the user identifier which is to be used by the message channel agent for authorization to access WebSphere MQ resources, including (if *PutAuthority* is MQPA_DEFAULT) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

This user identifier can be overridden by one supplied by a channel security exit.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SDR, MQCHT_SVR, MQCHT_CLNTCONN, MQCHT_CLUSSDR.

The maximum length of the MCA user identifier depends on the environment in which the MCA is running. MQ_MCA_USER_ID_LENGTH gives the maximum length for the environment for which your application is running. MQ_MAX_MCA_USER_ID_LENGTH gives the maximum for all supported environments.

On Windows , you can optionally qualify a user identifier with the domain name in the following format:

```
user@domain
```

MessageCompression **(MQCFIL)**
Header data compression techniques supported by the channel (parameter identifier: MQIACH_MSG_COMPRESSION). The list of message data compression techniques supported by the channel. For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference with the first compression technique supported by the remote end of the channel being used.

The mutually supported compression techniques of the channel are passed to the message exit of the sending channel where the compression technique used can be altered on a per message basis. Compression alters the data passed to send and receive exits.

Specify one or more of:

MQCOMPRESS_NONE
No message data compression is performed. This value is the default value.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_ANY
Any compression technique supported by the queue manager can be used. This value is only valid for receiver, requester, and server-connection channels.

ModeName **(MQCFST)**
Mode name (parameter identifier: MQCACH_MODE_NAME).

This parameter is the LU 6.2 mode name.

The maximum length of the string is MQ_MODE_NAME_LENGTH.

- On IBM i , HP Integrity NonStop Server, UNIX systems, and Windows , this parameter can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows) from the CPI-C symbolic destination name properties.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver or server-connection channels.

MsgExit **(MQCFSL)**
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately after a message has been retrieved from the transmission queue. The exit is given the entire application message and message descriptor for modification.

For channels with a channel type (*ChannelType*) of MQCHT_SVRCONN or MQCHT_CLNTCONN, this parameter is accepted but ignored, since message exits are not invoked for such channels.

The format of the string is the same as for *SecurityExit* .

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

You can specify a list of exit names by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.
- On z/OS , you can specify the names of up to eight exit programs.

MsgRetryCount **(MQCFIN)**
Message retry count (parameter identifier: MQIACH_MR_COUNT).

Specifies the number of times that a failing message must be retried.

Specify a value in the range 0 - 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

MsgRetryExit **(MQCFST)**
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

If a nonblank name is defined, the exit is invoked before performing a wait before retrying a failing message.

The format of the string is the same as for *SecurityExit* .

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

MsgRetryInterval **(MQCFIN)**
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

Specifies the minimum time interval in milliseconds between retries of failing messages.

Specify a value in the range 0 - 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

MsgRetryUserData **(MQCFST)**
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

Specifies user data that is passed to the message retry exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

MsgUserData **(MQCFSL)**
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

Specifies user data that is passed to the message exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

For channels with a channel type (*ChannelType*) of MQCHT_SVRCONN or MQCHT_CLNTCONN, this parameter is accepted but ignored, since message exits are not invoked for such channels.

You can specify a list of exit user data strings by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the *MsgExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/OS , you can specify up to eight strings.

NetworkPriority **(MQCFIN)**
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

The priority for the network connection. If there are multiple paths available, distributed queuing selects the path with the highest priority.

The value must be in the range 0 (lowest) - 9 (highest).

This parameter applies only to channels with a *ChannelType* of MQCHT_CLUSRCVR

NonPersistentMsgSpeed **(MQCFIN)**
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

This parameter is supported in the following environments: AIX , HP-UX, IBM i , Solaris, Windows, and Linux .

Specifying MQNPMS_FAST means that nonpersistent messages on a channel need not wait for a syncpoint before being made available for retrieval. The advantage of this is that nonpersistent messages become available for retrieval far more quickly. The disadvantage is that because they do not wait for a syncpoint, they might be lost if there is a transmission failure.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR. The value can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password **(MQCFST)**
Password (parameter identifier: MQCACH_PASSWORD).

This parameter is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent. On IBM i , HP Integrity NonStop Server, and UNIX systems, it is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, or MQCHT_CLUSSDR. On z/OS , it is valid only for a *ChannelType* value of MQCHT_CLNTCONN.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

PropertyControl **(MQCFIN)**
Property control attribute (parameter identifier MQIA_PROPERTY_CONTROL).

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor). The value can be:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of **mcd.** , **jms.** , **usr.** or **mqext.** , all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those properties contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This value is the default value; it allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message, except those properties in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

This attribute is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

PutAuthority **(MQCFIN)**
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

Specifies whether the user identifier in the context information associated with a message must be used to establish authority to put the message on the destination queue.

This parameter is valid only for channels with a *ChannelType* value of MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSRCVR, or MQCHT_SVRCONN.

The value can be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used. This value is not valid for channels of type MQCHT_SVRCONN.

MQPA_ALTERNATE_OR_MCA
The user ID from the *UserIdentifier* field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS and is not valid for channels of type MQCHT_SVRCONN.

MQPA_ONLY_MCA
The default user ID is used. Any user ID received from the network is not used. This value is supported only on z/OS .

QMgrName **(MQCFST)**
Queue-manager name (parameter identifier: MQCA_Q_MGR_NAME).

For channels with a *ChannelType* of MQCHT_CLNTCONN, this name is the name of a queue manager to which a client application can request connection.

For channels of other types, this parameter is not valid. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToChannelName* object (for Copy) or *ChannelName* object (for Create).
MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command. If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero: `DEFINE CHANNEL(channel-name) CHLTYPE(type) REPLACE QSGDISP(COPY)` The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.	The object definition resides in the shared repository. This definition is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero: `DEFINE CHANNEL(channe-name) CHLTYPE(type) REPLACE QSGDISP` The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

ReceiveExit **(MQCFSL)**

Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

If a nonblank name is defined, the exit is invoked before data received from the network is processed. The complete transmission buffer is passed to the exit and the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit* .

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

You can specify a list of exit names by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.
- On z/OS , you can specify the names of up to eight exit programs.

ReceiveUserData **(MQCFSL)**
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

Specifies user data that is passed to the receive exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

You can specify a list of exit user data strings by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the *ReceiveExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/OS , you can specify up to eight strings.

Replace **(MQCFIN)**
Replace channel definition (parameter identifier: MQIACF_REPLACE).

The value can be:

MQRP_YES
Replace existing definition.

If *ChannelType* is MQCHT_CLUSSDR, MQRP_YES can be specified only if the channel was created manually.

MQRP_NO
Do not replace existing definition.

SecurityExit **(MQCFST)**
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

If a nonblank name is defined, the security exit is invoked at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is enabled to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote machine are passed to the exit.

The exit is given the entire application message and message descriptor for modification.

The format of the string depends on the platform, as follows:

- On IBM i and UNIX systems, it is of the form

```
libraryname(functionname)
```

Note: On IBM i systems, the following form is also supported for compatibility with older releases:
```
progname libname
```

where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary).

- On Windows, it is of the form
```
dllname(functionname)
```

where *dllname* is specified without the suffix `.DLL`.
- On z/OS, it is a load module name, maximum length 8 characters (128 characters are allowed for exit names for client-connection channels, subject to a maximum total length of 999).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

SecurityUserData **(MQCFST)**
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

Specifies user data that is passed to the security exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit **(MQCFSL)**
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately before data is sent out on the network. The exit is given the complete transmission buffer before it is transmitted; the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit* .

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

You can specify a list of exit names by using an MQCFSL structure instead of an MQCFST structure.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.
- On z/OS , you can specify the names of up to eight exit programs.

SendUserData **(MQCFSL)**
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

Specifies user data that is passed to the send exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

You can specify a list of exit user data strings by using an MQCFSL structure instead of an MQCFST structure.

- Each exit user data string is passed to the exit at the same ordinal position in the *SendExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.

- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.
- On z/OS , you can specify up to eight strings.

SeqNumberWrap **(MQCFIN)**
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

Specifies the maximum message sequence number. When the maximum is reached, sequence numbers wrap to start again at 1.

The maximum message sequence number is not negotiable; the local and remote channels must wrap at the same number.

Specify a value in the range 100 - 999 999 999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

SharingConversations **(MQCFIN)**
Maximum number of sharing conversations (parameter identifier: MQIACH_SHARING_CONVERSATIONS).

Specifies the maximum number of conversations that can share a particular TCP/IP MQI channel instance (socket).

Specify a value in the range 0 - 999 999 999. The default value is 10 and the migrated value is 10.

This parameter is valid only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN. It is ignored for channels with a *TransportType* other than MQXPT_TCP.

The number of shared conversations does not contribute to the *MaxInstances* or *MaxInstancesPerClient* totals.

A value of:

1 Means that there is no sharing of conversations over a TCP/IP channel instance, but client heartbeating is available whether in an MQGET call or not, read ahead and client asynchronous consumption are available, and channel quiescing is more controllable.

0 Specifies no sharing of conversations over a TCP/IP channel instance. The channel instance runs in a mode before that of WebSphere MQ Version 7.0, regarding:
- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Client asynchronous consumption

ShortRetryCount **(MQCFIN)**
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

The maximum number of attempts that are made by a sender or server channel to establish a connection to the remote machine, at intervals specified by *ShortRetryInterval* before the (normally longer) *LongRetryCount* and *LongRetryInterval* are used.

Retry attempts are made if the channel fails to connect initially (whether it is started automatically by the channel initiator or by an explicit command), and also if the connection fails after the channel has successfully connected. However, if the cause of the failure is such that retry is unlikely to be successful, retries are not attempted.

Specify a value in the range 0 - 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

ShortRetryInterval **(MQCFIN)**
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

Specifies the short retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine.

The time is approximate.

Specify a value in the range 0 - 999 999. Values exceeding this value are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

SSLCipherSpec **(MQCFST)**
CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

It is valid only for channels with a transport type (TRPTYPE) of TCP. If the TRPTYPE is not TCP, the data is ignored and no error message is issued.

The SSLCIPH values must specify the same CipherSpec on both ends of the channel.

Specify the name of the CipherSpec that you are using. Alternatively, on IBM i , and z/OS , you can specify the two-digit hexadecimal code.

The following table shows the CipherSpecs that can be used with WebSphere MQ SSL.

On IBM i , installation of AC3 is a prerequisite of the use of SSL.

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
NULL_MD5 [a]	SSL 3.0	MD5	None	0	No	No	No
NULL_SHA [a]	SSL 3.0	SHA-1	None	0	No	No	No
RC4_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC4	40	No	No	No
RC4_MD5_US [a]	SSL 3.0	MD5	RC4	128	No	No	No
RC4_SHA_US [a]	SSL 3.0	SHA-1	RC4	128	No	No	No
RC2_MD5_EXPORT [2] [a]	SSL 3.0	MD5	RC2	40	No	No	No
DES_SHA_EXPORT [2] [a]	SSL 3.0	SHA-1	DES	56	No	No	No
RC4_56_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	RC4	56	No	No	No
DES_SHA_EXPORT1024 [3] [b]	SSL 3.0	SHA-1	DES	56	No	No	No
TLS_RSA_WITH_AES_128_CBC_SHA [a]	TLS 1.0	SHA-1	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA [4] [a]	TLS 1.0	SHA-1	AES	256	Yes	No	No
TLS_RSA_WITH_DES_CBC_SHA [a]	TLS 1.0	SHA-1	DES	56	No[5]	No	No
FIPS_WITH_DES_CBC_SHA [b]	SSL 3.0	SHA-1	DES	56	No[6]	No	No
TLS_RSA_WITH_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
TLS_RSA_WITH_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
TLS_RSA_WITH_AES_256_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	256	Yes	No	No
ECDHE_ECDSA_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No
ECDHE_RSA_RC4_128_SHA256 [b]	TLS 1.2	SHA_1	RC4	128	No	No	No
ECDHE_ECDSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_ECDSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_RSA_AES_128_CBC_SHA256 [b]	TLS 1.2	SHA-256	AES	128	Yes	No	No
ECDHE_RSA_AES_256_CBC_SHA384 [b]	TLS 1.2	SHA-384	AES	256	Yes	No	No
ECDHE_ECDSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	Yes	No
ECDHE_ECDSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	Yes
ECDHE_RSA_AES_128_GCM_SHA256 [b]	TLS 1.2	AEAD AES-128 GCM	AES	128	Yes	No	No
ECDHE_RSA_AES_256_GCM_SHA384 [b]	TLS 1.2	AEAD AES-256 GCM	AES	256	Yes	No	No
TLS_RSA_WITH_NULL_SHA256 [b]	TLS 1.2	SHA-256	None	0	No	No	No
ECDHE_RSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No
ECDHE_ECDSA_NULL_SHA256 [b]	TLS 1.2	SHA-1	None	0	No	No	No
TLS_RSA_WITH_NULL_NULL [b]	TLS 1.2	None	None	0	No	No	No
TLS_RSA_WITH_RC4_128_SHA256 [b]	TLS 1.2	SHA-1	RC4	128	No	No	No

A table describing the CipherSpecs you can use with WebSphere MQ SSL and TLS support.

CipherSpec name	Protocol used	Data integrity	Encryption algorithm	Encryption bits	FIPS[1]	Suite B 128 bit	Suite B 192 bit
Notes: 1. Specifies whether the CipherSpec is FIPS-certified on a FIPS-certified platform. See Federal Information Processing Standards (FIPS) for an explanation of FIPS. 2. The maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake. 3. The handshake key size is 1024 bits. 4. This CipherSpec cannot be used to secure a connection from the WebSphere MQ Explorer to a queue manager unless the appropriate unrestricted policy files are applied to the JRE used by the Explorer. 5. This CipherSpec was FIPS 140-2 certified before 19 May 2007. 6. This CipherSpec was FIPS 140-2 certified before 19 May 2007. The name FIPS_WITH_DES_CBC_SHA is historical and reflects the fact that this CipherSpec was previously (but is no longer) FIPS-compliant. This CipherSpec is deprecated and its use is not recommended. 7. This CipherSpec can be used to transfer up to 32 GB of data before the connection is terminated with error AMQ9288. To avoid this error, either avoid using triple DES, or enable secret key reset when using this CipherSpec. **Platform support:** • a Available on all supported platforms. • b Available only on UNIX, Linux, and Windows platforms.							

When you request a personal certificate, you specify a key size for the public and private key pair. The key size that is used during the SSL handshake can depend on the size stored in the certificate and on the CipherSpec:

- On UNIX systems, Windows systems, and z/OS , when a CipherSpec name includes _EXPORT , the maximum handshake key size is 512 bits. If either of the certificates exchanged during the SSL handshake has a key size greater than 512 bits, a temporary 512-bit key is generated for use during the handshake.
- On UNIX and Windows systems, when a CipherSpec name includes _EXPORT1024 , the handshake key size is 1024 bits.
- Otherwise the handshake key size is the size stored in the certificate.

If the SSLCIPH parameter is blank, no attempt is made to use SSL on the channel.

SSLClientAuth **(MQCFIN)**
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Client authentication required.

MQSCA_OPTIONAL
Client authentication optional.

Defines whether IBM WebSphere MQ requires a certificate from the SSL client.

The SSL client is the end of the message channel that initiates the connection. The SSL Server is the end of the message channel that receives the initiation flow.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

SSLPeerName **(MQCFST)**
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

Note: An alternative way of restricting connections into channels by matching against the SSL or TLS Subject Distinguished Name, is to use channel authentication records. With channel authentication records, different SSL or TLS Subject Distinguished Name patterns can be applied to the same channel. If both SSLPEER on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns in order to connect. For more information, see Channel authentication records.

On platforms other than z/OS , the length of the string is MQ_SSL_PEER_NAME_LENGTH. On z/OS , it is MQ_SSL_SHORT_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the SSL certificate.) If the Distinguished Name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

This parameter is optional; if it is not specified, the Distinguished Name of the peer is not checked when the channel is started. (The Distinguished Name from the certificate is still written into the SSLPEER definition held in memory, and passed to the security exit). If SSLCIPH is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The SSLPEER value is specified in the standard form used to specify a Distinguished Name. For example: `SSLPEER('SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN="H1_C_FR1",O=IBM,C=GB')`

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

Attribute	Description
SERIALNUMBER	Certificate serial number
MAIL	Email address
E	Email address (Deprecated in preference to MAIL)
UID or USERID	User identifier
CN	Common Name
T	Title
OU	Organizational Unit name
DC	Domain component
O	Organization name
STREET	Street / First line of address
L	Locality name
ST (or SP or S)	State or Province name
PC	Postal code / zip code
C	Country
UNSTRUCTUREDNAME	Host name
UNSTRUCTUREDADDRESS	IP address
DNQ	Distinguished name qualifier

WebSphere MQ only accepts uppercase letters for the attribute types.

If any of the unsupported attribute types are specified in the SSLPEER string, an error is output either when the attribute is defined or at run time (depending on which platform you are running on), and the string is deemed not to have matched the Distinguished Name of the flowed certificate.

If the Distinguished Name of the flowed certificate contains multiple OU (organizational unit) attributes, and SSLPEER specifies these attributes to be compared, they must be defined in descending hierarchical order. For example, if the Distinguished Name of the flowed certificate contains the OUs `OU=Large Unit,OU=Medium Unit,OU=Small Unit` , specifying the following SSLPEER values work:

```
('OU=Large Unit,OU=Medium Unit') ('OU=*,OU=Medium Unit,OU=Small Unit') ('OU=*,OU=Medium Unit')
```

but specifying the following SSLPEER values fail:

```
('OU=Medium Unit,OU=Small Unit') ('OU=Large Unit,OU=Small Unit') ('OU=Medium Unit')
```

Any or all the attribute values can be generic, either an asterisk (*) on its own, or a stem with initiating or trailing asterisks. This value allows the SSLPEER to match any Distinguished Name value, or any value starting with the stem for that attribute.

If an asterisk is specified at the beginning or end of any attribute value in the Distinguished Name on the certificate, you can specify * to check for an exact match in SSLPEER. For example, if you have an attribute of CN=Test* in the Distinguished Name of the certificate, you can use the following command:

```
SSLPEER('CN=Test\*')
```

TpName **(MQCFST)**

Transaction program name (parameter identifier: MQCACH_TP_NAME).

This name is the LU 6.2 transaction program name.

The maximum length of the string is MQ_TP_NAME_LENGTH.

- On IBM i, HP Integrity NonStop Server, UNIX systems, and Windows , this parameter can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows) from the CPI-C symbolic destination name properties.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver channels.

TransportType **(MQCFIN)**

Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

This value is supported in Windows . It also applies to z/OS for defining client-connection channels that connect to servers on the platforms supporting NetBIOS.

MQXPT_SPX
SPX.

This value is supported in Windows. It also applies to z/OS for defining client-connection channels that connect to servers on the platforms supporting SPX.

UseDLQ **(MQCFIN)**

Determines whether the dead-letter queue is used when messages cannot be delivered by channels. (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value can be:

MQUSEDLQ_NO
Messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the NonPersistentMsgSpeed setting.

MQUSEDLQ_YES
When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used, else the behavior is as for MQUSEDLQ_NO.

UserIdentifier **(MQCFST)**
Task user identifier (parameter identifier: MQCACH_USER_ID).

This parameter is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent. On IBM i and UNIX systems, it is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR. On z/OS , it is valid only for a *ChannelType* value of MQCHT_CLNTCONN.

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

XmitQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

A transmission queue name is required (either previously defined or specified here) if *ChannelType* is MQCHT_SENDER or MQCHT_SERVER. It is not valid for other channel types.

Error codes (Change, Copy, and Create Channel)

This command might return the following error codes in the response format header, in addition to those codes listed in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_BATCH_INT_ERROR
Batch interval not valid.

MQRCCF_BATCH_INT_WRONG_TYPE
Batch interval parameter not allowed for this channel type.

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

MQRCCF_HB_INTERVAL_ERROR
Heartbeat interval not valid.

MQRCCF_HB_INTERVAL_WRONG_TYPE
Heartbeat interval parameter not allowed for this channel type.

MQRCCF_LONG_RETRY_ERROR
Long retry count not valid.

MQRCCF_LONG_RETRY_WRONG_TYPE
Long retry parameter not allowed for this channel type.

MQRCCF_LONG_TIMER_ERROR
Long timer not valid.

MQRCCF_LONG_TIMER_WRONG_TYPE
Long timer parameter not allowed for this channel type.

MQRCCF_MAX_INSTANCES_ERROR
Maximum instances value not valid.

MQRCCF_MAX_INSTS_PER_CLNT_ERR
Maximum instances per client value not valid.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_MCA_NAME_ERROR
Message channel agent name error.

MQRCCF_MCA_NAME_WRONG_TYPE
Message channel agent name not allowed for this channel type.

MQRCCF_MCA_TYPE_ERROR
Message channel agent type not valid.

MQRCCF_MISSING_CONN_NAME
Connection name parameter required but missing.

MQRCCF_MR_COUNT_ERROR
Message retry count not valid.

MQRCCF_MR_COUNT_WRONG_TYPE
Message-retry count parameter not allowed for this channel type.

MQRCCF_MR_EXIT_NAME_ERROR
Channel message-retry exit name error.

MQRCCF_MR_EXIT_NAME_WRONG_TYPE
Message-retry exit parameter not allowed for this channel type.

MQRCCF_MR_INTERVAL_ERROR
Message retry interval not valid.

MQRCCF_MR_INTERVAL_WRONG_TYPE
Message-retry interval parameter not allowed for this channel type.

MQRCCF_MSG_EXIT_NAME_ERROR
Channel message exit name error.

MQRCCF_NET_PRIORITY_ERROR
Network priority value error.

MQRCCF_NET_PRIORITY_WRONG_TYPE
Network priority attribute not allowed for this channel type.

MQRCCF_NPM_SPEED_ERROR
Nonpersistent message speed not valid.

MQRCCF_NPM_SPEED_WRONG_TYPE
Nonpersistent message speed parameter not allowed for this channel type.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PUT_AUTH_ERROR
Put authority value not valid.

MQRCCF_PUT_AUTH_WRONG_TYPE
Put authority parameter not allowed for this channel type.

MQRCCF_RCV_EXIT_NAME_ERROR
Channel receive exit name error.

MQRCCF_SEC_EXIT_NAME_ERROR
Channel security exit name error.

MQRCCF_SEND_EXIT_NAME_ERROR
Channel send exit name error.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHARING_CONVS_ERROR
Value given for Sharing Conversations not valid.

MQRCCF_SHARING_CONVS_TYPE
Sharing Conversations parameter not valid for this channel type.

MQRCCF_SHORT_RETRY_ERROR
Short retry count not valid.

MQRCCF_SHORT_RETRY_WRONG_TYPE
Short retry parameter not allowed for this channel type.

MQRCCF_SHORT_TIMER_ERROR
Short timer value not valid.

MQRCCF_SHORT_TIMER_WRONG_TYPE
Short timer parameter not allowed for this channel type.

MQRCCF_SSL_CIPHER_SPEC_ERROR
SSL CipherSpec not valid.

MQRCCF_SSL_CLIENT_AUTH_ERROR
SSL client authentication not valid.

MQRCCF_SSL_PEER_NAME_ERROR
SSL peer name not valid.

MQRCCF_WRONG_CHANNEL_TYPE
Parameter not allowed for this channel type.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Change, Copy, and Create Channel (MQTT):

The Change Channel command changes existing Telemetry channel definitions. The Copy and Create Channel commands create new Telemetry channel definitions - the Copy command uses attribute values of an existing channel definition.

The Change Channel (MQCMD_CHANGE_CHANNEL) command changes the specified attributes in a channel definition. For any optional parameters that are omitted, the value does not change.

The Copy Channel (MQCMD_COPY_CHANNEL) command creates new channel definition using, for attributes not specified in the command, the attribute values of an existing channel definition.

The Create Channel (MQCMD_CREATE_CHANNEL) command creates a WebSphere MQ channel definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager. If a system default channel exists for the type of channel being created, the default values are taken from there.

Required parameters (Change, Create Channel)

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies the name of the channel definition to be changed, or created

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required on all types of channel; on a CLUSSDR it can be different from on the other channel types. If your convention for naming channels includes the name of the queue manager, you can make a CLUSSDR definition using the +QMNAME+ construction, and WebSphere MQ substitutes the correct repository queue manager name in place of +QMNAME+. This facility applies to AIX , HP-UX, Linux, IBM i, Solaris, and Windows only. See Configuring a queue manager cluster for more details.

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed, copied, or created. The value can be:

MQCHT_MQTT
Telemetry.

TrpType **(MQCFIN)**
Transmission protocol type of the channel (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE). This parameter is required for a create command in telemetry.

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value is:

MQXPT_TCP
TCP.

Port **(MQCFIN)**
The port number to use if *TrpType* is set to MQXPT_TCP. This parameter is required for a create command in telemetry, if *TrpType* is set to MQXPT_TCP.

The value is in the range 1 - 65335.

Required parameters (Copy Channel)

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed, copied, or created. The value can be:

MQCHT_MQTT
Telemetry.

Optional parameters (Change, Copy, and Create Channel)

Backlog **(MQCFIN)**
The number of concurrent connection requests that the telemetry channel supports at any one time (parameter identifier: MQIACH_BACKLOG).

The value is in the range 0 - 999999999.

JAASConfig **(MQCFST)**
The file path of the JAAS configuration (parameter identifier: MQCACH_JAAS_CONFIG).

The maximum length of this value is MQ_JAAS_CONFIG_LENGTH.

Only one of JAASCONFIG, MCAUSER, and USECLIENTID can be specified for a telemetry channel; if none is specified, no authentication is performed. If JAASConfig is specified, the client flows a user name and password. In all other cases, the flowed user name is ignored.

LocalAddress **(MQCFST)**
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

The value that you specify depends on the transport type *(TransportType)* to be used:

TCP/IP
The value is the optional IP address and optional port or port range to be used for outbound TCP/IP communications. The format for this information is as follows:

```
[ip-addr][(low-port[,high-port])]
```

where `ip-addr` is specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric form, and `low-port` and `high-port` are port numbers enclosed in parentheses. All are optional.

All Others
The value is ignored; no error is diagnosed.

Use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. This parameter is useful when a machine is connected to multiple networks with different IP addresses.

Examples of use

Value	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98 (1000)	Channel binds to this address and port 1000 locally
9.20.4.98 (1000,2000)	Channel binds to this address and uses a port in the range 1000 - 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to a port in the range 1000 - 2000 locally

Note:

- Do not confuse this parameter with *ConnectionName* . The *LocalAddress* parameter specifies the characteristics of the local communications; the *ConnectionName* parameter specifies how to reach a remote queue manager.

SSLCipherSuite **(MQCFST)**
CipherSuite (parameter identifier: MQCACH_SSL_CIPHER_SUITE).

The length of the string is MQ_SSL_CIPHER_SUITE_LENGTH.

SSL CIPHER SUITE character channel parameter type.

SSLClientAuth **(MQCFIN)**
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Client authentication required.

MQSCA_OPTIONAL
Client authentication optional.

Defines whether IBM WebSphere MQ requires a certificate from the SSL client.

The SSL client is the end of the message channel that initiates the connection. The SSL Server is the end of the message channel that receives the initiation flow.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

SSLKeyFile **(MQCFST)**
The store for digital certificates and their associated private keys (parameter identifier: MQCA_SSL_KEY_REPOSITORY).

If you do not specify a key file, SSL is not used.

The maximum length of this parameter is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLPassPhrase **(MQCFST)**
The password for the key repository (parameter identifier: MQCACH_SSL_KEY_PASSPHRASE).

If no pass phrase is entered, then unencrypted connections must be used.

The maximum length of this parameter is MQ_SSL_KEY_PASSPHRASE_LENGTH.

TransportType **(MQCFIN)**
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

This value is supported in Windows.

MQXPT_SPX
SPX.

This value is supported in Windows.

This parameter is required for a create command in telemetry; see TransportType for more information.

UseClientIdentifier **(MQCFIN)**
Determines whether to use the client ID of a new connection as the user ID for that connection (parameter identifier: MQIACH_USE_CLIENT_ID).

The value is either:

MQUCI_YES
Yes.

MQUCI_NO
No.

Only one of JAASCONFIG, MCAUSER, and USECLIENTID can be specified for a telemetry channel; if none is specified, no authentication is performed. If USECLIENTID is specified, the flowed user name of the client is ignored.

Error codes (Change, Copy, and Create Channel)

This command might return the following error codes in the response format header, in addition to those codes listed in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_BATCH_INT_ERROR
Batch interval not valid.

MQRCCF_BATCH_INT_WRONG_TYPE
Batch interval parameter not allowed for this channel type.

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

MQRCCF_HB_INTERVAL_ERROR
Heartbeat interval not valid.

MQRCCF_HB_INTERVAL_WRONG_TYPE
Heartbeat interval parameter not allowed for this channel type.

MQRCCF_LONG_RETRY_ERROR
Long retry count not valid.

MQRCCF_LONG_RETRY_WRONG_TYPE
Long retry parameter not allowed for this channel type.

MQRCCF_LONG_TIMER_ERROR
Long timer not valid.

MQRCCF_LONG_TIMER_WRONG_TYPE
Long timer parameter not allowed for this channel type.

MQRCCF_MAX_INSTANCES_ERROR
Maximum instances value not valid.

MQRCCF_MAX_INSTS_PER_CLNT_ERR
Maximum instances per client value not valid.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_MCA_NAME_ERROR
Message channel agent name error.

MQRCCF_MCA_NAME_WRONG_TYPE
Message channel agent name not allowed for this channel type.

MQRCCF_MCA_TYPE_ERROR
Message channel agent type not valid.

MQRCCF_MISSING_CONN_NAME
Connection name parameter required but missing.

MQRCCF_MR_COUNT_ERROR
Message retry count not valid.

MQRCCF_MR_COUNT_WRONG_TYPE
Message-retry count parameter not allowed for this channel type.

MQRCCF_MR_EXIT_NAME_ERROR
Channel message-retry exit name error.

MQRCCF_MR_EXIT_NAME_WRONG_TYPE
Message-retry exit parameter not allowed for this channel type.

MQRCCF_MR_INTERVAL_ERROR
Message retry interval not valid.

MQRCCF_MR_INTERVAL_WRONG_TYPE
Message-retry interval parameter not allowed for this channel type.

MQRCCF_MSG_EXIT_NAME_ERROR
Channel message exit name error.

MQRCCF_NET_PRIORITY_ERROR
Network priority value error.

MQRCCF_NET_PRIORITY_WRONG_TYPE
Network priority attribute not allowed for this channel type.

MQRCCF_NPM_SPEED_ERROR
Nonpersistent message speed not valid.

MQRCCF_NPM_SPEED_WRONG_TYPE
Nonpersistent message speed parameter not allowed for this channel type.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PUT_AUTH_ERROR
Put authority value not valid.

MQRCCF_PUT_AUTH_WRONG_TYPE
Put authority parameter not allowed for this channel type.

MQRCCF_RCV_EXIT_NAME_ERROR
Channel receive exit name error.

MQRCCF_SEC_EXIT_NAME_ERROR
Channel security exit name error.

MQRCCF_SEND_EXIT_NAME_ERROR
Channel send exit name error.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHARING_CONVS_ERROR
Value given for Sharing Conversations not valid.

MQRCCF_SHARING_CONVS_TYPE
Sharing Conversations parameter not valid for this channel type.

MQRCCF_SHORT_RETRY_ERROR
Short retry count not valid.

MQRCCF_SHORT_RETRY_WRONG_TYPE
Short retry parameter not allowed for this channel type.

MQRCCF_SHORT_TIMER_ERROR
Short timer value not valid.

MQRCCF_SHORT_TIMER_WRONG_TYPE
Short timer parameter not allowed for this channel type.

MQRCCF_SSL_CIPHER_SPEC_ERROR
SSL CipherSpec not valid.

MQRCCF_SSL_CLIENT_AUTH_ERROR
SSL client authentication not valid.

MQRCCF_SSL_PEER_NAME_ERROR
SSL peer name not valid.

MQRCCF_WRONG_CHANNEL_TYPE
Parameter not allowed for this channel type.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Change, Copy, and Create Channel Listener:

The Change Channel Listener command changes existing channel listener definitions. The Copy and Create Channel Listener commands create new channel listener definitions - the Copy command uses attribute values of an existing channel listener definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

The Change Channel Listener (MQCMD_CHANGE_LISTENER) command changes the specified attributes of an existing WebSphere MQ listener definition. For any optional parameters that are omitted, the value does not change.

The Copy Channel Listener (MQCMD_ COPY_LISTENER) command creates a WebSphere MQ listener definition, using, for attributes not specified in the command, the attribute values of an existing listener definition.

The Create Channel Listener (MQCMD_CREATE_LISTENER) command creates a WebSphere MQ listener definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameters (Change and Create Channel Listener)

ListenerName **(MQCFST)**
The name of the listener definition to be changed or created (parameter identifier: MQCACH_LISTENER_NAME).

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

TransportType **(MQCFIN)**
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_TCP
TCP.

MQXPT_LU62
LU 6.2. This value is valid only on Windows.

MQXPT_NETBIOS
NetBIOS. This value is valid only on Windows.

MQXPT_SPX
SPX. This value is valid only on Windows.

Required parameters (Copy Channel Listener)

FromListenerName **(MQCFST)**
The name of the listener definition to be copied from (parameter identifier: MQCACF_FROM_LISTENER_NAME).

This parameter specifies the name of the existing listener definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

ToListenerName **(MQCFST)**
To listener name (parameter identifier: MQCACF_TO_LISTENER_NAME).

This parameter specifies the name of the new listener definition. If a listener definition with this name exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters (Change, Copy, and Create Channel Listener)

Adapter **(MQCFIN)**
Adapter number (parameter identifier: MQIACH_ADAPTER).

The adapter number on which NetBIOS listens. This parameter is valid only on Windows.

Backlog **(MQCFIN)**
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

Commands **(MQCFIN)**
Adapter number (parameter identifier: MQIACH_COMMAND_COUNT).

The number of commands that the listener can use. This parameter is valid only on Windows.

IPAddress **(MQCFST)**
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form. If you do not specify a value for this parameter, the listener listens on all configured IPv4 and IPv6 stacks.

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH

ListenerDesc **(MQCFST)**
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

This parameter is a plain-text comment that provides descriptive information about the listener definition. It must contain only displayable characters.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

LocalName **(MQCFST)**
NetBIOS local name (parameter identifier: MQCACH_LOCAL_NAME).

The NetBIOS local name that the listener uses. This parameter is valid only on Windows.

The maximum length of the string is MQ_CONN_NAME_LENGTH

NetbiosNames **(MQCFIN)**
NetBIOS names (parameter identifier: MQIACH_NAME_COUNT).

The number of names that the listener supports. This parameter is valid only on Windows.

Port **(MQCFIN)**
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP. This parameter is valid only if the value of *TransportType* is MQXPT_TCP.

Replace **(MQCFIN)**
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a namelist definition with the same name as *ToListenerName* exists, this definition specifies whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

Sessions **(MQCFIN)**
NetBIOS sessions (parameter identifier: MQIACH_SESSION_COUNT).

The number of sessions that the listener can use. This parameter is valid only on Windows.

Socket **(MQCFIN)**
SPX socket number (parameter identifier: MQIACH_SOCKET).

The SPX socket on which to listen. This parameter is valid only if the value of *TransportType* is MQXPT_SPX.

StartMode **(MQCFIN)**
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. This value is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The listener is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

TPName **(MQCFST)**
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The LU 6.2 transaction program name. This parameter is valid only on Windows.

The maximum length of the string is MQ_TP_NAME_LENGTH

Change, Copy, and Create Communication Information Object:

The Change Communication Information Object command changes existing communication information object definitions. The Copy and Create Communication Information Object commands create new communication information object definitions - the Copy command uses attribute values of an existing communication information object definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

The Change communication information (MQCMD_CHANGE_COMM_INFO) command changes the specified attributes of an existing WebSphere MQ communication information object definition. For any optional parameters that are omitted, the value does not change.

The Copy communication information (MQCMD_COPY_COMM_INFO) command creates a WebSphere MQ communication information object definition, using, for attributes not specified in the command, the attribute values of an existing communication information definition.

The Create communication information (MQCMD_CREATE_COMM_INFO) command creates a WebSphere MQ communication information object definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change communication information)

ComminfoName **(MQCFST)**
The name of the communication information definition to be changed (parameter identifier: MQCA_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Required parameters (Copy communication information)

FromComminfoName **(MQCFST)**
The name of the communication information object definition to be copied from (parameter identifier: MQCACF_FROM_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

ToComminfoName **(MQCFST)**
The name of the communication information definition to copy to (parameter identifier: MQCACF_TO_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Required parameters (Create communication information)

CominfoName **(MQCFST)**
The name of the communication information definition to be created (parameter identifier: MQCA_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Optional parameters (Change, Copy, and Create communication information)

Bridge **(MQCFIN)**
Controls whether publications from applications not using Multicast are bridged to applications using multicast (parameter identifier: MQIA_MCAST_BRIDGE).

Bridging does not apply to topics that are marked as **MCAST(ONLY)**. As these topics can only have multicast traffic, it is not applicable to bridge to the non-multicast publish/subscribe domain.

MQMCB_DISABLED
Publications from applications not using multicast are not bridged to applications that do use Multicast. This is the default for IBM i.

MQMCB_ENABLED
Publications from applications not using multicast are bridged to applications that do use Multicast. This is the default for platforms other than IBM i. This value is not valid on IBM i.

CCSID **(MQCFIN)**
The coded character set identifier that messages are transmitted on (parameter identifier: MQIA_CODED_CHAR_SET_ID).

Specify a value in the range 1 to 65535.

The CCSID must specify a value that is defined for use on your platform, and use a character set that is appropriate to the platform. If you use this parameter to change the CCSID, applications that are running when the change is applied continue to use the original CCSID. Because of this, you must stop and restart all running applications before you continue.

This includes the command server and channel programs. To do this, stop and restart the queue manager after making the change. The default value is ASPUB which means that the coded character set is taken from the one that is supplied in the published message.

CommEvent **(MQCFIN)**
Controls whether event messages are generated for multicast handles that are created using this COMMINFO object (parameter identifier: MQIA_COMM_EVENT).

Events are only generated if monitoring is also enabled using the *MonitorInterval* parameter.

MQEVR_DISABLED
Publications from applications not using multicast are not bridged to applications that do use multicast. This is the default value.

MQEVR_ENABLED
Publications from applications not using multicast are bridged to applications that do use multicast.

MQEVR_EXCEPTION
Event messages are written if the message reliability is below the reliability threshold The reliability threshold is set to 90 by default.

Description **(MQCFST)**
Plain-text comment that provides descriptive information about the communication information object (parameter identifier: MQCA_COMM_INFO_DESC).

It must contain only displayable characters. The maximum length is 64 characters. In a DBCS installation, it can contain DBCS characters (subject to a maximum length of 64 bytes).

If characters are used that are not in the coded character set identifier (CCSID) for this queue manager, they might be translated incorrectly if the information is sent to another queue manager.

The maximum length is MQ_COMM_INFO_DESC_LENGTH.

Encoding **(MQCFIN)**
The encoding that the messages are transmitted in (parameter identifier: MQIACF_ENCODING).

MQENC_AS_PUBLISHED
The encoding of the message is taken from the one that is supplied in the published message. This is the default value.

MQENC_NORMAL

MQENC_REVERSED

MQENC_S390

MQENC_TNS

GrpAddress **(MQCFST)**
The group IP address or DNS name (parameter identifier: MQCACH_GROUP_ADDRESS).

It is the administrator's responsibility to manage the group addresses. It is possible for all multicast clients to use the same group address for every topic; only the messages that match outstanding subscriptions on the client are delivered. Using the same group address can be inefficient because every client must examine and process every multicast packet in the network. It is more efficient to allocate different IP group addresses to different topics or sets of topics, but this requires careful management, especially if other non-MQ multicast applications are in use on the network. The default value is `239.0.0.0`.

The maximum length is MQ_GROUP_ADDRESS_LENGTH.

MonitorInterval **(MQCFIN)**
How frequently monitoring information is updated and event messages are generated (parameter identifier: MQIA_MONITOR_INTERVAL).

The value is specified as a number of seconds in the range 0 to 999 999. A value of `0` indicates that no monitoring is required.

If a non-zero value is specified, monitoring is enabled. Monitoring information is updated and event messages (if enabled using *CommEvent*, are generated about the status of the multicast handles created using this communication information object.

MsgHistory **(MQCFIN)**
This value is the amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs (parameter identifier: MQIACH_MSG_HISTORY).

The value is in the range 0 to 999 999 999. A value of `0` gives the least level of reliability. The default value is `100`.

MulticastHeartbeat **(MQCFIN)**
The heartbeat interval is measured in milliseconds, and specifies the frequency at which the transmitter notifies any receivers that there is no further data available (parameter identifier: MQIACH_MC_HB_INTERVAL).

The value is in the range 0 to 999 999. The default value is `2000` milliseconds.

MulticastPropControl **(MQCFIN)**
The multicast properties control how many of the MQMD properties and user properties flow with the message (parameter identifier: MQIACH_MULTICAST_PROPERTIES).

MQMCP_ALL
All user properties and all the fields of the MQMD are transported. This is the default value.

MQMCP_REPLY
Only user properties, and MQMD fields that deal with replying to the messages, are transmitted. These properties are:

- MsgType
- MessageId
- CorrelId
- ReplyToQ
- ReplyToQmgr

MQMCP_USER
Only the user properties are transmitted.

MQMCP_NONE
No user properties or MQMD fields are transmitted.

MQMCP_COMPAT
Properties are transmitted in a format compatible with previous MQ multicast clients.

NewSubHistory **(MQCFIN)**
The new subscriber history controls whether a subscriber joining a publication stream receives as much data as is currently available, or receives only publications made from the time of the subscription (parameter identifier: MQIACH_NEW_SUBSCRIBER_HISTORY).

MQNSH_NONE
A value of NONE causes the transmitter to transmit only publication made from the time of the subscription. This is the default value.

MQNSH_ALL
A value of ALL causes the transmitter to retransmit as much history of the topic as is known. In some circumstances, this can give a similar behavior to retained publications.

Using the value of MQNSH_ALL might have a detrimental effect on performance if there is a large topic history because all the topic history is retransmitted.

PortNumber **(MQCFIN)**
The port number to transmit on (parameter identifier: MQIACH_PORT).

The default port number is `1414`.

Type **(MQCFIN)**
The type of the communications information object (parameter identifier: MQIA_COMM_INFO_TYPE).

The only type supported is `MQCIT_MULTICAST`.

Change, Copy, and Create Namelist:

The Change Namelist command changes existing namelist definitions. The Copy and Create Namelist commands create new namelist definitions - the Copy command uses attribute values of an existing namelist definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

The Change Namelist (MQCMD_CHANGE_NAMELIST) command changes the specified attributes of an existing WebSphere MQ namelist definition. For any optional parameters that are omitted, the value does not change.

The Copy Namelist (MQCMD_COPY_NAMELIST) command creates a WebSphere MQ namelist definition, using, for attributes not specified in the command, the attribute values of an existing namelist definition.

The Create Namelist (MQCMD_CREATE_NAMELIST) command creates a WebSphere MQ namelist definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change and Create Namelist)

NamelistName **(MQCFST)**
The name of the namelist definition to be changed (parameter identifier: MQCA_NAMELIST_NAME).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Required parameters (Copy Namelist)

FromNamelistName **(MQCFST)**
The name of the namelist definition to be copied from (parameter identifier: MQCACF_FROM_NAMELIST_NAME).

This parameter specifies the name of the existing namelist definition that contains values for the attributes not specified in this command.

On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToNamelistName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

ToNamelistName **(MQCFST)**
To namelist name (parameter identifier: MQCACF_TO_NAMELIST_NAME).

This parameter specifies the name of the new namelist definition. If a namelist definition with this name exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters (Change, Copy, and Create Namelist)

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

NamelistDesc **(MQCFST)**
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

This parameter is a plain-text comment that provides descriptive information about the namelist definition. It must contain only displayable characters.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

NamelistType **(MQCFIN)**
Type of names in the namelist (parameter identifier: MQIA_NAMELIST_TYPE). This parameter applies to z/OS only.

Specifies the type of names in the namelist. The value can be:

MQNT_NONE
The names are of no particular type.

MQNT_Q
A namelist that holds a list of queue names.

MQNT_CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

MQNT_AUTH_INFO
The namelist is associated with SSL, and contains a list of authentication information object names.

Names **(MQCFSL)**
The names to be placed in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the *Count* field in the MQCFSL structure. The length of each name is given by the *StringLength* field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToNameListName* object (for Copy) or *NameListName* object (for Create).

QSGDisposition	Change	Copy, Create

MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.
	
	If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they refresh local copies on page set zero:
	`DEFINE NAMELIST(name)`
	`REPLACE QSGDISP(COPY)`
	
	The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
		The object definition resides in the shared repository. This is allowed only if the queue manager is in a queue-sharing group.
		
		If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they make or refresh local copies on page set zero:
		`DEFINE NAMELIST(name)`
		`REPLACE QSGDISP(COPY)`
		
		The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

Replace **(MQCFIN)**
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a namelist definition with the same name as *ToNamelistName* exists, this definition specifies whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

Change, Copy, and Create Process:

The Change Process command changes existing process definitions. The Copy and Create Process commands create new process definitions - the Copy command uses attribute values of an existing process definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

The Change Process (MQCMD_CHANGE_PROCESS) command changes the specified attributes of an existing WebSphere MQ process definition. For any optional parameters that are omitted, the value does not change.

The Copy Process (MQCMD_COPY_PROCESS) command creates a WebSphere MQ process definition, using, for attributes not specified in the command, the attribute values of an existing process definition.

The Create Process (MQCMD_CREATE_PROCESS) command creates a WebSphere MQ process definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameters (Change and Create Process)

ProcessName **(MQCFST)**
The name of the process definition to be changed or created (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Required parameters (Copy Process)

FromProcessName **(MQCFST)**
The name of the process definition to be copied from (parameter identifier: MQCACF_FROM_PROCESS_NAME).

Specifies the name of the existing process definition that contains values for the attributes not specified in this command.

On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToProcessName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

ToProcessName **(MQCFST)**
To process name (parameter identifier: MQCACF_TO_PROCESS_NAME).

The name of the new process definition. If a process definition with this name exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters (Change, Copy, and Create Process)

ApplId **(MQCFST)**
Application identifier (parameter identifier: MQCA_APPL_ID).

ApplId is the name of the application to be started. The application must be on the platform for which the command is executing. The name might typically be a fully qualified file name of an executable object. Qualifying the file name is particularly important if you have multiple IBM WebSphere MQ installations, to ensure the correct version of the application is run.

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

ApplType **(MQCFIN)**
Application type (parameter identifier: MQIA_APPL_TYPE).

Valid application types are:

MQAT_OS400
IBM i application.

MQAT_WINDOWS_NT
Windows or Windows 95, Windows 98 application.

MQAT_DOS
DOS client application.

MQAT_WINDOWS
Windows client application.

MQAT_UNIX
UNIX application.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_CICS
CICS transaction.

MQAT_NSK
HP Integrity NonStop Server application.

MQAT_ZOS
z/OS application.

MQAT_DEFAULT
Default application type.

integer: System-defined application type in the range zero through 65 535 or a user-defined application type in the range 65 536 through 999 999 999 (not checked).

Only specify application types (other than user-defined types) that are supported on the platform at which the command is executed:

- On IBM i:

  MQAT_OS400,
  MQAT_CICS, and
  MQAT_DEFAULT are supported.
- On HP Integrity NonStop Server:

  MQAT_NSK,
  MQAT_DOS,
  MQAT_WINDOWS, and
  MQAT_DEFAULT are supported.
- On UNIX systems:

  MQAT_UNIX,
  MQAT_OS2,
  MQAT_DOS,
  MQAT_WINDOWS,
  MQAT_CICS, and
  MQAT_DEFAULT are supported.
- On Windows:

  MQAT_WINDOWS_NT,
  MQAT_OS2,
  MQAT_DOS,
  MQAT_WINDOWS,
  MQAT_CICS, and
  MQAT_DEFAULT are supported.
- On z/OS:

MQAT_DOS,
MQAT_IMS
MQAT_MVS,
MQAT_UNIX,
MQAT_CICS, and
MQAT_DEFAULT are supported.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. In a shared queue environment, you can provide a different queue manager name from the one you are using to enter the command. The command server must be enabled.
- An asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

EnvData **(MQCFST)**
Environment data (parameter identifier: MQCA_ENV_DATA).

A character string that contains environment information pertaining to the application to be started.

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc **(MQCFST)**
Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

A plain-text comment that provides descriptive information about the process definition. It must contain only displayable characters.

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToProcessName* object (for Copy) or *ProcessName* object (for Create).

QSGDisposition	Change	Copy, Create

MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameters QSGDISP(GROUP). On the page set of the queue manager that executes the command, only a local copy of the object is altered by this command. If the command is successful, the following command is generated.
	`DEFINE PROCESS(process-name)`
	`REPLACE QSGDISP(COPY)`
	
	The command is sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero. The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
		The object definition resides in the shared repository. `GROUP` is allowed only if the queue manager is in a queue-sharing group. If the definition is successful, the following command is generated.
		`DEFINE PROCESS(process-name)`
		`REPLACE QSGDISP(COPY)`
		
		The command is sent to all active queue managers in the queue-sharing group to attempt to make or refresh local copies on page set zero. The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. MQQSGD_Q_MGR is the default value.	The object is defined on the page set of the queue manager that executes the command. MQQSGD_Q_MGR is the default value.

Replace **(MQCFIN)**
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a process definition with the same name as *ToProcessName* exists, specify whether to replace it.

The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

UserData **(MQCFST)**
User data (parameter identifier: MQCA_USER_DATA).

A character string that contains user information pertaining to the application (defined by *ApplId*) that is to be started.

For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Change, Copy, and Create Queue:

The Change Queue command changes existing queue definitions. The Copy and Create Queue commands create new queue definitions - the Copy command uses attribute values of an existing queue definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
✓	✓	✓

The Change Queue command MQCMD_CHANGE_Q changes the specified attributes of an existing WebSphere MQ queue. For any optional parameters that are omitted, the value does not change.

The Copy Queue command MQCMD_COPY_Q creates a queue definition of the same type. For attributes not specified in the command, it uses the attribute values of an existing queue definition.

The Create Queue command MQCMD_CREATE_Q creates a queue definition with the specified attributes. All attributes that are not specified are set to the default value for the type of queue that is created.

Required parameters (Change and Create Queue)

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be changed. The maximum length of the string is MQ_Q_NAME_LENGTH.

Required parameters (Copy Queue)

FromQName **(MQCFST)**
From queue name (parameter identifier: MQCACF_FROM_Q_NAME).

Specifies the name of the existing queue definition.

On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR, MQQSGD_COPY, or MQQSGD_SHARED to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToQName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_Q_NAME_LENGTH.

ToQName **(MQCFST)**
To queue name (parameter identifier: MQCACF_TO_Q_NAME).

Specifies the name of the new queue definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue names must be unique; if a queue definition exists with the name and type of the new queue, *Replace* must be specified as MQRP_YES. If a queue definition exists with the same name as and a different type from the new queue, the command fails.

Required parameters (all commands)

QType **(MQCFIN)**
Queue type (parameter identifier: MQIA_Q_TYPE).

The value specified must match the type of the queue being changed.

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

Optional parameters (Change, Copy, and Create Queue)

BackoutRequeueName **(MQCFST)**
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

Specifies the name of the queue to which a message is transferred if it is backed out more times than the value of *BackoutThreshold*. The queue does not have to be a local queue.

The backout queue does not need to exist at this time but it must exist when the *BackoutThreshold* value is exceeded.

The maximum length of the string is MQ_Q_NAME_LENGTH.

BackoutThreshold **(MQCFIN)**
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

The number of times a message can be backed out before it is transferred to the backout queue specified by *BackoutRequeueName*.

If the value is later reduced, messages that are already on the queue that were backed out at least as many times as the new value remain on the queue. Those messages are transferred if they are backed out again.

Specify a value in the range 0 - 999,999,999.

BaseObjectName **(MQCFST)**
Name of the object to which the alias resolves (parameter identifier: MQCA_BASE_OBJECT_NAME).

This parameter is the name of a queue or topic that is defined to the local queue manager.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

BaseQName **(MQCFST)**
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This parameter is the name of a local or remote queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CFStructure **(MQCFST)**
coupling facility structure name (parameter identifier: MQCA_CF_STRUC_NAME). This parameter applies to z/OS only.

Specifies the name of the coupling facility structure where you want to store messages when you use shared queues. The name:

- Cannot have more than 12 characters
- Must start with an uppercase letter (A - Z)
- Can include only the characters A - Z and 0 - 9

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

The name of the queue-sharing group to which the queue manager is connected is prefixed to the name you supply. The name of the queue-sharing group is always four characters, padded with @ symbols if necessary. For example, if you use a queue-sharing group named NY03 and you supply the

name PRODUCT7, the resultant coupling facility structure name is NY03PRODUCT7. Note the administrative structure for the queue-sharing group (in this case NY03CSQ_ADMIN) cannot be used for storing messages.

For local and model queues, the following rules apply. The rules apply if you use the Create Queue command with a value of MQRP_YES in the *Replace* parameter. The rules also apply if you use the Change Queue command.

- On a local queue with a value of MQQSGD_SHARED in the *QSGDisposition* parameter, *CFStructure* cannot change.

  If you need to change either the *CFStructure* or *QSGDisposition* value, you must delete and redefine the queue. To preserve any of the messages on the queue you must offload the messages before you delete the queue. Reload the messages after you redefine the queue, or move the messages to another queue.
- On a model queue with a value of MQQDT_SHARED_DYNAMIC in the *DefinitionType* parameter, *CFStructure* cannot be blank.
- On a local queue with a value other than MQQSGD_SHARED in the *QSGDisposition* parameter, the value of *CFStructure* does not matter. The value *CFStructure* also does not matter for a model queue with a value other than MQQDT_SHARED_DYNAMIC in the *DefinitionType* parameter.

For local and model queues, when you use the Create Queue command with a value of MQRP_NO in the *Replace* parameter, the coupling facility structure:

- On a local queue with a value of MQQSGD_SHARED in the *QSGDisposition* parameter, or a model queue with a value of MQQDT_SHARED_DYNAMIC in the *DefinitionType* parameter, *CFStructure* cannot be blank.
- On a local queue with a value other than MQQSGD_SHARED in the *QSGDisposition* parameter, the value of *CFStructure* does not matter. The value *CFStructure* also does not matter for a model queue with a value other than MQQDT_SHARED_DYNAMIC in the *DefinitionType* parameter.

Note: Before you can use the queue, the structure must be defined in the coupling facility Resource Management (CFRM) policy data set.

ClusterChannelName **(MQCFST)**

This parameter is supported only on transmission queues.

ClusterChannelName is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue. ClusterChannelName is not supported on z/OS. (Parameter identifier: MQCA_CLUS_CHL_NAME.)

You can also set the transmission queue attribute ClusterChannelName attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the ClusterChannelName attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either SYSTEM.CLUSTER.TRANSMIT.*ChannelName* or SYSTEM.CLUSTER.TRANSMIT.QUEUE, depending on the value of the queue manager DefClusterXmitQueueType attribute.

By specifying asterisks, "*", in ClusterChannelName, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. ClusterChannelName is limited to a length of 20 characters: MQ_CHANNEL_NAME_LENGTH.

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. The default configuration can be changed by modified by changing the queue manager attribute, DefClusterXmitQueueType. The default value

of the attribute is SCTQ. You can change the value to CHANNEL. If you set the DefClusterXmitQueueType attribute to CHANNEL, each cluster-sender channel defaults to using a specific cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to which the queue belongs.

Changes to this parameter do not affect instances of the queue that are open.

Only one of the resultant values of *ClusterName* and *ClusterNamelist* can be nonblank; you cannot specify a value for both.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist **(MQCFST)**
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist, that specifies a list of clusters to which the queue belongs.

Changes to this parameter do not affect instances of the queue that are open.

Only one of the resultant values of *ClusterName* and *ClusterNamelist* can be nonblank; you cannot specify a value for both.

CLWLQueuePriority **(MQCFIN)**
Cluster workload queue priority (parameter identifier: MQIA_CLWL_Q_PRIORITY).

Specifies the priority of the queue in cluster workload management; see Configuring a queue manager cluster . The value must be in the range 0 - 9, where 0 is the lowest priority and 9 is the highest.

CLWLQueueRank **(MQCFIN)**
Cluster workload queue rank (parameter identifier: MQIA_CLWL_Q_RANK).

Specifies the rank of the queue in cluster workload management. The value must be in the range 0 - 9, where 0 is the lowest priority and 9 is the highest.

CLWLUseQ **(MQCFIN)**
Cluster workload use remote queue (parameter identifier: MQIA_CLWL_USEQ).

Specifies whether remote and local queues are to be used in cluster workload distribution. The value can be:

MQCLWL_USEQ_AS_Q_MGR
Use the value of the *CLWLUseQ* parameter on the definition of the queue manager.

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is run when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank, or omit the parameter altogether. The command is run on the queue manager on which it was entered.
- A queue manager name. The command is run on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.

- An asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Custom **(MQCFST)**
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes are named. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

This description is updated when features using this attribute are introduced. There are currently no values for *Custom*.

DefaultPutResponse **(MQCFIN)**
Default put response type definition (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The parameter specifies the type of response to be used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF. The value can be:

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

DefBind **(MQCFIN)**
Bind definition (parameter identifier: MQIA_DEF_BIND).

The parameter specifies the binding to be used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call. The value can be:

MQBND_BIND_ON_OPEN
The binding is fixed by the MQOPEN call.

MQBND_BIND_NOT_FIXED
The binding is not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

Changes to this parameter do not affect instances of the queue that are open.

DefinitionType **(MQCFIN)**
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

The value can be:

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_SHARED_DYNAMIC
Dynamically defined shared queue. This option is available on z/OS only.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

DefInputOpenOption **(MQCFIN)**
Default input open option (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

Specifies the default share option for applications opening this queue for input.

The value can be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

DefPersistence **(MQCFIN)**
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

Specifies the default for message-persistence on the queue. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value can be:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority **(MQCFIN)**
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Specifies the default priority of messages put on the queue. The value must be in the range zero through to the maximum priority value that is supported (9).

DefReadAhead **(MQCFIN)**
Default read ahead (parameter identifier: MQIA_DEF_READ_AHEAD).

Specifies the default read ahead behavior for non-persistent messages delivered to the client.

The value can be:

MQREADA_NO
Non-persistent messages are not read ahead unless the client application is configured to request read ahead.

MQREADA_YES
Non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages it is sent.

MQREADA_DISABLED
Read ahead of non-persistent messages is not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

DistLists **(MQCFIN)**
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

Specifies whether distribution-list messages can be placed on the queue.

Note: This attribute is set by the sending message channel agent (MCA). The sending MCA removes messages from the queue each time it establishes a connection to a receiving MCA on a partner queue manager. The attribute is not normally set by administrators, although it can be set if the need arises.

This parameter is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, and Linux.

The value can be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

Force **(MQCFIN)**
Force changes (parameter identifier: MQIACF_FORCE).

Specifies whether the command must be forced to complete when conditions are such that completing the command would affect an open queue. The conditions depend upon the type of the queue that is being changed:

QALIAS *BaseQName* is specified with a queue name and an application has the alias queue open.

QLOCAL Either of the following conditions indicates that a local queue would be affected:

- *Shareability* is specified as MQQA_NOT_SHAREABLE and more than one application has the local queue open for input.
- The *Usage* value is changed and one or more applications has the local queue open, or there are one or more messages on the queue. (The *Usage* value must not normally be changed while there are messages on the queue. The format of messages changes when they are put on a transmission queue.)

QREMOTE
Either of the following conditions indicates that a remote queue would be affected:

- If *XmitQName* is specified with a transmission-queue name, or blank, and an application has a remote queue open that would be affected by this change.
- If any of the following parameters are specified with a queue or queue-manager name, and one or more applications has a queue open that resolved through this definition as a queue-manager alias. The parameters are:
  1. *RemoteQName*
  2. *RemoteQMgrName*
  3. *XmitQName*

QMODEL This parameter is not valid for model queues.

Note: A value of MQFC_YES is not required if this definition is in use as a reply-to queue definition only.
The value can be:

MQFC_YES
Force the change.

MQFC_NO
Do not force the change.

HardenGetBackout **(MQCFIN)**
Harden the backout count, or not (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

Specifies whether the count of backed out messages is saved (hardened) across restarts of the message queue manager.

Note: WebSphere MQ for IBM i always hardens the count, regardless of the setting of this attribute.

The value can be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count might not be remembered.

IndexType **(MQCFIN)**
Index type (parameter identifier: MQIA_INDEX_TYPE). This parameter applies to z/OS only.

Specifies the type of index maintained by the queue manager to expedite MQGET operations on the queue. For shared queues, the type of index determines what type of MQGET calls can be used. The value can be:

MQIT_NONE
No index.

MQIT_MSG_ID
The queue is indexed using message identifiers.

MQIT_CORREL_ID
The queue is indexed using correlation identifiers.

MQIT_MSG_TOKEN
The queue is indexed using message tokens.

MQIT_GROUP_ID
The queue is indexed using group identifiers.

Messages can be retrieved using a selection criterion only if an appropriate index type is maintained, as the following table shows:

Retrieval selection criterion	*IndexType* **required**	
	Shared queue	**Other queue**
None (sequential retrieval)	Any	Any
Message identifier	MQIT_MSG_ID or MQIT_NONE	Any
Correlation identifier	MQIT_CORREL_ID	Any
Message and correlation identifiers	MQIT_MSG_ID or MQIT_CORREL_ID	Any
Group identifier	MQIT_GROUP_ID	Any
Grouping	MQIT_GROUP_ID	MQIT_GROUP_ID
Message token	Not allowed	MQIT_MSG_TOKEN

InhibitGet **(MQCFIN)**
Get operations are allowed or inhibited (parameter identifier: MQIA_INHIBIT_GET).

The value can be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut **(MQCFIN)**
Put operations are allowed or inhibited (parameter identifier: MQIA_INHIBIT_PUT).

Specifies whether messages can be put on the queue.

The value can be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

InitiationQName **(MQCFST)**
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The local queue for trigger messages relating to this queue. The initiation queue must be on the same queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

The maximum length for messages on the queue. Applications might use the value of this attribute to determine the size of buffer they need to retrieve messages from the queue. If you change this value it might cause an application to operate incorrectly.

Do not set a value that is greater than the *MaxMsgLength* attribute of a queue manager.

The lower limit for this parameter is 0. The upper limit depends on the environment:

- On AIX, HP Integrity NonStop Server, HP-UX, IBM i, Solaris, Linux, Windows, and z/OS, the maximum message length is 100 MB (104,857,600 bytes).
- On other UNIX systems, the maximum message length is 4 MB (4,194,304 bytes).

MaxQDepth **(MQCFIN)**
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

The maximum number of messages allowed on the queue.

Note: Other factors might cause the queue to be treated as full. For example, it appears to be full if there is no storage available for a message.

Specify a value greater than or equal to 0, and less than or equal to:

- 999,999,999 if the queue is on AIX, HP-UX, IBM i, Solaris, Linux, Windows, or z/OS
- 640,000 if the queue is on any otherIBM WebSphere MQ platform.

MsgDeliverySequence **(MQCFIN)**
Messages are delivered in priority order or sequence (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value can be:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

NonPersistentMessageClass **(MQCFIN)**
The level of reliability to be assigned to non-persistent messages that are put to the queue (parameter identifier: MQIA_NPM_CLASS).

The value can be:

MQNPM_CLASS_NORMAL
Non-persistent messages persist as long as the lifetime of the queue manager session. They are discarded in the event of a queue manager restart. This value is the default value.

MQNPM_CLASS_HIGH
The queue manager attempts to retain non-persistent messages for the lifetime of the queue. Non-persistent messages might still be lost in the event of a failure.

This parameter is valid only on local and model queues. It is not valid on z/OS.

ProcessName **(MQCFST)**
Name of process definition for the queue (parameter identifier: MQCA_PROCESS_NAME).

Specifies the local name of the WebSphere MQ process that identifies the application to be started when a trigger event occurs.

- If the queue is a transmission queue, the process definition contains the name of the channel to be started. This parameter is optional for transmission queues on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS. If you do not specify it, the channel name is taken from the value specified for the *TriggerData* parameter.
- In other environments, the process name must be nonblank for a trigger event to occur, although it can be set after creating the queue.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

PropertyControl **(MQCFIN)**
Property control attribute (parameter identifier: MQIA_PROPERTY_CONTROL).

Specifies how message properties are handled when messages are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option. The value can be:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those properties contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This value is the default value. It allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message are removed from the message before the message is sent to the remote queue manager. Properties in the message descriptor, or extension, are not removed.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

MQPROP_FORCE_MQRFH2
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible using the message handle.

MQPROP_V6COMPAT
Any application MQRFH2 header is received as it was sent. Any properties set using MQSETMP must be retrieved using MQINQMP. They are not added to the MQRFH2 created by the application. Properties that were set in the MQRFH2 header by the sending application cannot be retrieved using MQINQMP.

This parameter is applicable to Local, Alias, and Model queues.

QDepthHighEvent **(MQCFIN)**
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

A Queue Depth High event indicates that an application put a message on a queue. This event caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the *QDepthHighLimit* parameter.

Note: The value of this attribute can change implicitly; see "Definitions of the Programmable Command Formats" on page 796.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighLimit **(MQCFIN)**
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth High event.

This event indicates that an application put a message to a queue. This event caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the *QDepthHighEvent* parameter.

The value is expressed as a percentage of the maximum queue depth, *MaxQDepth*. It must be greater than or equal to 0 and less than or equal to 100.

QDepthLowEvent **(MQCFIN)**
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

A Queue Depth Low event indicates that an application retrieved a message from a queue. This event caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the *QDepthLowLimit* parameter.

Note: The value of this attribute can change implicitly. See "Definitions of the Programmable Command Formats" on page 796.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowLimit **(MQCFIN)**
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This event indicates that an application retrieved a message from a queue. This event caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the *QDepthLowEvent* parameter.

Specify the value as a percentage of the maximum queue depth (*MaxQDepth* attribute), in the range 0 through 100.

QDepthMaxEvent **(MQCFIN)**
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

A Queue Full event indicates that an MQPUT call to a queue was rejected because the queue is full. That is, the queue depth reached its maximum value.

Note: The value of this attribute can change implicitly; see "Definitions of the Programmable Command Formats" on page 796.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDesc **(MQCFST)**
Queue description (parameter identifier: MQCA_Q_DESC).

Text that briefly describes the object.

The maximum length of the string is MQ_Q_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing. This choice ensures that the text is translated correctly if it is sent to another queue manager.

QServiceInterval **(MQCFIN)**
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events. See the *QServiceIntervalEvent* parameter.

Specify a value in the range 0 through 999 999 999 milliseconds.

QServiceIntervalEvent **(MQCFIN)**
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

A Queue Service Interval High event is generated when a check indicates that no messages were retrieved from, or put to, the queue for at least the time indicated by the *QServiceInterval* attribute.

A Queue Service Interval OK event is generated when a check indicates that a message was retrieved from the queue within the time indicated by the *QServiceInterval* attribute.

Note: The value of this attribute can change implicitly; see "Definitions of the Programmable Command Formats" on page 796.

The value can be:

MQQSIE_HIGH
Queue Service Interval High events enabled.
- Queue Service Interval High events are enabled and
- Queue Service Interval OK events are disabled.

MQQSIE_OK
Queue Service Interval OK events enabled.
- Queue Service Interval High events are disabled and
- Queue Service Interval OK events are enabled.

MQQSIE_NONE
No queue service interval events enabled.
- Queue Service Interval High events are disabled and
- Queue Service Interval OK events are also disabled.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command using the MQQSGD_GROUP object of the same name as the *ToQName* object (for Copy) or the *QName* object (for Create). For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.

MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.
	
	If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to attempt to refresh local copies on page set zero:
	`DEFINE QUEUE(q-name)`
	`REPLACE QSGDISP(COPY)`
	
	The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
		The object definition resides in the shared repository. This value is allowed only in a shared queue manager environment.
		
		If the definition is successful, the following MQSC command is generated and sent to all active queue managers to attempt to make or refresh local copies on page set zero:
		`DEFINE QUEUE(q-name)`
		`REPLACE QSGDISP(COPY)`
		
		The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value. For local queues, messages are stored on the page sets of each queue manager and are available only through that queue manager.
MQQSGD_SHARED	This value applies only to local queues. The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_SHARED. Any object residing on the page set of the queue manager that executes the command, or any object defined by a command using the parameter MQQSGD_GROUP, is not affected by this command.	This option applies only to local queues. The object is defined in the shared repository. Messages are stored in the coupling facility and are available to any queue manager in the queue-sharing group. You can specify MQQSGD_SHARED only if: • *CFStructure* is nonblank • *IndexType* is not MQIT_MSG_TOKEN • The queue is not one of the following: – SYSTEM.CHANNEL.INITQ – SYSTEM.COMMAND.INPUT

QueueAccounting **(MQCFIN)**

Controls the collection of accounting data (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_Q_MGR
The collection of accounting data for the queue is performed based upon the setting of the *QueueAccounting* parameter on the queue manager.

MQMON_OFF
Accounting data collection is disabled for the queue.

MQMON_ON
If the value of the queue manager's *QueueAccounting* parameter is not MQMON_NONE, accounting data collection is enabled for the queue.

QueueMonitoring **(MQCFIN)**
Online monitoring data collection (parameter identifier: MQIA_MONITORING_Q).

Specifies whether online monitoring data is to be collected and, if so, the rate at which the data is collected. The value can be:

MQMON_OFF
Online monitoring data collection is turned off for this queue.

MQMON_Q_MGR
The value of the queue manager's *QueueMonitoring* parameter is inherited by the queue.

MQMON_LOW
If the value of the queue manager *QueueMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on. The rate of data collection is low for this queue.

MQMON_MEDIUM
If the value of the queue manager *QueueMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on. The rate of data collection is moderate for this queue.

MQMON_HIGH
If the value of the queue manager *QueueMonitoring* parameter is not MQMON_NONE, online monitoring data collection is turned on. The rate of data collection is high for this queue.

QueueStatistics **(MQCFIN)**
Statistics data collection (parameter identifier: MQIA_STATISTICS_Q).

Specifies whether statistics data collection is enabled. The value can be:

MQMON_Q_MGR
The value of the queue manager's *QueueStatistics* parameter is inherited by the queue.

MQMON_OFF
Statistics data collection is disabled

MQMON_ON
If the value of the queue manager's *QueueStatistics* parameter is not MQMON_NONE, statistics data collection is enabled

This parameter is valid only on IBM i, UNIX systems, and Windows.

RemoteQMgrName **(MQCFST)**
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank or the name of the queue manager the application is connected to. If *XmitQName* is blank there must be a local queue called *RemoteQMgrName*. That queue is used as the transmission queue.

If this definition is used for a queue-manager alias, *RemoteQMgrName* is the name of the queue manager. The queue manager name can be the name of the connected queue manager. If *XmitQName* is blank, when the queue is opened there must be a local queue called *RemoteQMgrName*. That queue is used as the transmission queue.

If this definition is used for a reply-to queue alias, *RemoteQMgrName* is the name of the queue manager that is to be the reply-to queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteQName **(MQCFST)**
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

If this definition is used for a local definition of a remote queue, *RemoteQName* must not be blank when the open occurs.

If this definition is used for a queue-manager alias definition, *RemoteQName* must be blank when the open occurs.

If this definition is used for a reply-to queue alias, this name is the name of the queue that is to be the reply-to queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Replace **(MQCFIN)**
Replace attributes (parameter identifier: MQIACF_REPLACE). This parameter is not valid on a Change Queue command.

If the object exists, the effect is like issuing the Change Queue command. It is like a Change Queue command without the MQFC_YES option on the *Force* parameter, and with all of the other attributes specified. In particular, note that any messages which are on the existing queue are retained.

The Change Queue command without MQFC_YES on the *Force* parameter, and the Create Queue command with MQRP_YES on the *Replace* parameter, are different. The difference is that the Change Queue command does not change unspecified attributes. Create Queue with MQRP_YES sets all the attributes. If you use MQRP_YES, unspecified attributes are taken from the default definition, and the attributes of the object being replaced, if one exists, are ignored.)

The command fails if both of the following are true:

- The command sets attributes that would require the use of MQFC_YES on the *Force* parameter if you were using the Change Queue command
- The object is open

The Change Queue command with MQFC_YES on the *Force* parameter succeeds in this situation.

If MQSCO_CELL is specified on the *Scope* parameter on UNIX systems, and there is already a queue with the same name in the cell directory, the command fails. The command fails even if MQRP_YES is specified.

The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

RetentionInterval **(MQCFIN)**
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

The number of hours for which the queue might be needed, based on the date and time when the queue was created.

This information is available to a housekeeping application or an operator and can be used to determine when a queue is no longer required. The queue manager does not delete queues nor does it prevent queues from being deleted if their retention interval is not expired. It is the responsibility of the user to take any required action.

Specify a value in the range 0 - 999,999,999.

Scope **(MQCFIN)**
Scope of the queue definition (parameter identifier: MQIA_SCOPE).

Specifies whether the scope of the queue definition extends beyond the queue manager which owns the queue. It does so if the queue name is contained in a cell directory, so that it is known to all the queue managers within the cell.

If this attribute is changed from MQSCO_CELL to MQSCO_Q_MGR, the entry for the queue is deleted from the cell directory.

Model and dynamic queues cannot be changed to have cell scope.

If it is changed from MQSCO_Q_MGR to MQSCO_CELL, an entry for the queue is created in the cell directory. If there is already a queue with the same name in the cell directory, the command fails. The command also fails if no name service supporting a cell directory is configured.

The value can be:

MQSCO_Q_MGR
Queue-manager scope.

MQSCO_CELL
Cell scope.

This value is not supported on IBM i.

This parameter is not available on z/OS.

Shareability **(MQCFIN)**
The queue can be shared, or not (parameter identifier: MQIA_SHAREABILITY).

Specifies whether multiple instances of applications can open this queue for input.

The value can be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

StorageClass **(MQCFST)**
Storage class (parameter identifier: MQCA_STORAGE_CLASS). This parameter applies to z/OS only.

Specifies the name of the storage class.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

TargetType **(MQCFIN)**
Target type (parameter identifier: MQIA_BASE_TYPE).

Specifies the type of object to which the alias resolves.

The value can be:

MQOT_Q The object is a queue.

MQOT_TOPIC
The object is a topic.

TriggerControl **(MQCFIN)**
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

Specifies whether trigger messages are written to the initiation queue.

The value can be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerData **(MQCFST)**
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

Specifies user data that the queue manager includes in the trigger message. This data is made available to the monitoring application that processes the initiation queue and to the application that is started by the monitor.

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

TriggerDepth **(MQCFIN)**
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

Specifies (when *TriggerType* is MQTT_DEPTH) the number of messages that initiates a trigger message to the initiation queue. The value must be in the range 1 through 999 999 999.

TriggerMsgPriority **(MQCFIN)**
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

Specifies the minimum priority that a message must have before it can cause, or be counted for, a trigger event. The value must be in the range of priority values that is supported (0 through 9).

TriggerType **(MQCFIN)**
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

Specifies the condition that initiates a trigger event. When the condition is true, a trigger message is sent to the initiation queue.

The value can be:

MQTT_NONE
No trigger messages.

MQTT_EVERY
Trigger message for every message.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

Usage **(MQCFIN)**
Usage (parameter identifier: MQIA_USAGE).

Specifies whether the queue is for normal usage or for transmitting messages to a remote message queue manager.

The value can be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

XmitQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

Specifies the local name of the transmission queue to be used for messages destined for either a remote queue or for a queue-manager alias definition.

If *XmitQName* is blank, a queue with the same name as *RemoteQMgrName* is used as the transmission queue.

This attribute is ignored if the definition is being used as a queue-manager alias and *RemoteQMgrName* is the name of the connected queue manager.

It is also ignored if the definition is used as a reply-to queue alias definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Error codes (Change, Copy, and Create Queue)

This command might return the following errors in the response format header, in addition to the values shown on in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CELL_DIR_NOT_AVAILABLE
Cell directory is not available.

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_CLUSTER_Q_USAGE_ERROR
Cluster usage conflict.

MQRCCF_DYNAMIC_Q_SCOPE_ERROR
Dynamic queue scope error.

MQRCCF_FORCE_VALUE_ERROR
Force value not valid.

MQRCCF_Q_ALREADY_IN_CELL
Queue exists in cell.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

Change Queue Manager:

The Change Queue Manager (MQCMD_CHANGE_Q_MGR) command changes the specified attributes of the queue manager.

HP Integrity NonStop Server	UNIX and Linux	Windows
✓	✓	✓

For any optional parameters that are omitted, the value does not change.

Required parameters:
None

Optional parameters (Change Queue Manager)

AccountingConnOverride **(MQCFIN)**
Specifies whether applications can override the settings of the *QueueAccounting* and *MQIAccounting* queue manager parameters (parameter identifier: MQIA_ACCOUNTING_CONN_OVERRIDE).

The value can be:

MQMON_DISABLED
Applications cannot override the settings of the *QueueAccounting* and *MQIAccounting* parameters.

This value is the initial default value for the queue manager.

MQMON_ENABLED
Applications can override the settings of the *QueueAccounting* and *MQIAccounting* parameters by using the options field of the MQCNO structure of the MQCONNX API call.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

AccountingInterval **(MQCFIN)**
The time interval, in seconds, at which intermediate accounting records are written (parameter identifier: MQIA_ACCOUNTING_INTERVAL).

Specify a value in the range 1 - 604,000.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ActivityRecording **(MQCFIN)**
Specifies whether activity reports can be generated (parameter identifier: MQIA_ACTIVITY_RECORDING).

The value can be:

MQRECORDING_DISABLED
Activity reports cannot be generated.

MQRECORDING_MSG
Activity reports can be generated and sent to the reply queue specified by the originator in the message causing the report.

MQRECORDING_Q
Activity reports can be generated and sent to SYSTEM.ADMIN.ACTIVITY.QUEUE.

AdoptNewMCACheck **(MQCFIN)**
The elements checked to determine whether an MCA must be adopted (restarted) when a new inbound channel is detected. It must be adopted (restarted) if it that has the same name as a currently active MCA (parameter identifier: MQIA_ADOPTNEWMCA_CHECK).

The value can be:

MQADOPT_CHECK_Q_MGR_NAME
Check the queue manager name.

MQADOPT_CHECK_NET_ADDR
Check the network address.

MQADOPT_CHECK_ALL
Check the queue manager name and network address. Perform this check to prevent your channels from being inadvertently shut down. This value is the initial default value of the queue manager.

MQADOPT_CHECK_NONE
Do not check any elements.

This parameter applies to z/OS only.

AdoptNewMCAType **(MQCFIN)**
Adoption of orphaned channel instances (parameter identifier: MQIA_ADOPTNEWMCA_TYPE).

Specify whether an orphaned MCA instance is to be adopted when a new inbound channel request is detected matching the *AdoptNewMCACheck* parameters.

The value can be:

MQADOPT_TYPE_NO
Do not adopt orphaned channel instances.

MQADOPT_TYPE_ALL
Adopt all channel types. This value is the initial default value of the queue manager.

This parameter applies to z/OS only.

AuthorityEvent **(MQCFIN)**
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled. This value is not permitted on z/OS.

BridgeEvent **(MQCFIN)**
Controls whether IMS Bridge events are generated (parameter identifier: MQIA_BRIDGE_EVENT). This parameter applies to z/OS only.

The value can be:

MQEVR_DISABLED
Event reporting disabled. This value is the default value.

MQEVR_ENABLED
Event reporting enabled. This value is not supported on z/OS.

CertificateValPolicy **(MQCFIN)**
Specifies which SSL/TLS certificate validation policy is used to validate digital certificates received from remote partner systems (parameter identifier: MQIA_CERT_VAL_POLICY).

This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards. For more information, see Certificate validation policies in WebSphere MQ.

The value can be:

MQ_CERT_VAL_POLICY_ANY
Apply each of the certificate validation policies supported by the secure sockets library and accept the certificate chain if any of the policies considers the certificate chain valid. This setting can be used for maximum backwards compatibility with older digital certificates which do not comply with the modern certificate standards.

MQ_CERT_VAL_POLICY_RFC5280
Apply only the RFC 5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

This parameter is only valid on UNIX, Linux, and Windows and can be used only on a queue manager with a command level of 711, or higher.

Changes to **CertificateValPolicy** become effective either:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.
- For channels that run as threads of a process pooling process, when the process pooling process is started or restarted and first runs an SSL channel. If the process pooling process has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**. The process pooling process is amqrmppa on UNIX, Linux, and Windows systems.
- When a **REFRESH SECURITY TYPE(SSL)** command is issued.

CFConlos **(MQCFIN)**
Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with CFConlos set to ASQMGR (parameter identifier: MQIA_QMGR_CFCONLOS).

The value can be:

MQCFCONLOS_TERMINATE
The queue manager terminates when connectivity to CF structures is lost.

MQCFCONLOS_TOLERATE
The queue manager tolerates loss of connectivity to CF structures without terminating.

This parameter applies to z/OS only.

You can select MQCFCONLOS_TOLERATE only if all the queue managers in the queue-sharing group are at command level 710 or greater and have OPMODE set to NEWFUNC.

ChannelAutoDef **(MQCFIN)**
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: MQIA_CHANNEL_AUTO_DEF).

Auto-definition for cluster-sender channels is always enabled.

This parameter is supported in the following environments: IBM i, UNIX, Linux, and Windows systems.

The value can be:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

ChannelAutoDefEvent **(MQCFIN)**
Controls whether channel auto-definition events are generated (parameter identifier: MQIA_CHANNEL_AUTO_DEF_EVENT), when a receiver, server-connection, or cluster-sender channel is auto-defined.

This parameter is supported in the following environments: IBM i, UNIX, Linux, and Windows systems.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDefExit **(MQCFIN)**
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

This exit is invoked when an inbound request for an undefined channel is received, if:

1. The channel is a cluster-sender, or
2. Channel auto-definition is enabled (see *ChannelAutoDef*).

This exit is also invoked when a cluster-receiver channel is started.

The format of the name is the same as for the *SecurityExit* parameter described in "Change, Copy, and Create Channel" on page 813.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is supported in the following environments: IBM i, z/OS, UNIX, Linux, and Windows. On z/OS, it applies only to cluster-sender and cluster-receiver channels.

ChannelAuthenticationRecords **(MQCFIN)**
Controls whether channel authentication records are used. Channel authentication records can still be set and displayed regardless of the value of this attribute. (parameter identifier: MQIA_CHLAUTH_RECORDS).

The value can be:

MQCHLA_DISABLED
Channel authentication records are not checked.

MQCHLA_ENABLED
Channel authentication records are checked.

ChannelEvent **(MQCFIN)**
Controls whether channel events are generated (parameter identifier: MQIA_CHANNEL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_EXCEPTION
Reporting of exception channel events enabled.

ChannelInitiatorControl **(MQCFIN)**
Specifies whether the channel initiator is to be started when the queue manager starts (parameter identifier: MQIA_CHINIT_CONTROL).

The value can be:

MQSVC_CONTROL_MANUAL
The channel initiator is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The channel initiator is to be started automatically when the queue manager starts.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ChannelMonitoring **(MQCFIN)**
Default setting for online monitoring for channels (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_NONE
Online monitoring data collection is turned off for channels regardless of the setting of their *ChannelMonitoring* parameter.

MQMON_OFF
Online monitoring data collection is turned off for channels specifying a value of MQMON_Q_MGR in their *ChannelMonitoring* parameter. This value is the initial default value of the queue manager.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelMonitoring* parameter.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelMonitoring* parameter.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelMonitoring* parameter.

ChannelStatistics **(MQCFIN)**
Controls whether statistics data is to be collected for channels (parameter identifier: MQIA_STATISTICS_CHANNEL).

The value can be:

MQMON_NONE
Statistics data collection is turned off for channels regardless of the setting of their *ChannelStatistics* parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection is turned off for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

MQMON_LOW
Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

MQMON_HIGH
Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ChinitAdapters **(MQCFIN)**
Number of adapter subtasks (parameter identifier: MQIA_CHINIT_ADAPTERS).

The number of adapter subtasks to use for processing IBM WebSphere MQ calls. This parameter applies to z/OS only.

Specify a value in the range 1 - 9999. The initial default value of the queue manager is 8.

ChinitDispatchers **(MQCFIN)**
Number of dispatchers (parameter identifier: MQIA_CHINIT_DISPATCHERS).

The number of dispatchers to use for the channel initiator. This parameter applies to z/OS only.

Specify a value in the range 1 - 9999. The initial default value of the queue manager is 5.

ChinitServiceParm **(MQCFIN)**
Reserved for use by IBM (parameter identifier: MQCA_CHINIT_SERVICE_PARM).

This parameter applies to z/OS only.

ChinitTraceAutoStart **(MQCFIN)**
Specifies whether the channel initiator trace must start automatically (parameter identifier: MQIA_CHINIT_TRACE_AUTO_START).

The value can be:

MQTRAXSTR_YES
Channel initiator trace is to start automatically.

MQTRAXSTR_NO
Channel initiator trace is not to start automatically. This value is the initial default value of the queue manager.

This parameter applies to z/OS only.

ChinitTraceTableSize **(MQCFIN)**
The size, in megabytes, of the trace data space of the channel initiator (parameter identifier: MQIA_CHINIT_TRACE_TABLE_SIZE).

Specify a value in the range 2 - 2048. The initial default value of the queue manager is 2.

This parameter applies to z/OS only.

ClusterSenderMonitoringDefault **(MQCFIN)**
Default setting for online monitoring for automatically defined cluster-sender channels (parameter identifier: MQIA_MONITORING_AUTO_CLUSSDR).

Specifies the value to be used for the *ChannelMonitoring* attribute of automatically defined cluster-sender channels. The value can be:

MQMON_Q_MGR
Collection of online monitoring data is inherited from the setting of the queue manager's *ChannelMonitoring* parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Monitoring for the channel is switched off.

MQMON_LOW
Unless *ChannelMonitoring* is MQMON_NONE, this value specifies a low rate of data collection with a minimal effect on system performance. The data collected is not likely to be the most current.

MQMON_MEDIUM
Unless *ChannelMonitoring* is MQMON_NONE, this value specifies a moderate rate of data collection with limited effect on system performance.

MQMON_HIGH
Unless *ChannelMonitoring* is MQMON_NONE, this value specifies a high rate of data collection with a likely effect on system performance. The data collected is the most current available.

ClusterSenderStatistics **(MQCFIN)**
Controls whether statistics data is to be collected for auto-defined cluster-sender channels (parameter identifier: MQIA_STATISTICS_AUTO_CLUSSDR).

The value can be:

MQMON_Q_MGR
Collection of statistics data is inherited from the setting of the queue manager's *ChannelStatistics* parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection for the channel is switched off.

MQMON_LOW
Unless *ChannelStatistics* is MQMON_NONE, this value specifies a low rate of data collection with a minimal effect on system performance.

MQMON_MEDIUM
Unless *ChannelStatistics* is MQMON_NONE, this value specifies a moderate rate of data collection.

MQMON_HIGH
Unless *ChannelStatistics* is MQMON_NONE, this value specifies a high rate of data collection.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ClusterWorkLoadData **(MQCFST)**
Cluster workload exit data (parameter identifier: MQCA_CLUSTER_WORKLOAD_DATA).

This parameter is passed to the cluster workload exit when it is called.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

ClusterWorkLoadExit **(MQCFST)**
Cluster workload exit name (parameter identifier: MQCA_CLUSTER_WORKLOAD_EXIT).

If a nonblank name is defined this exit is invoked when a message is put to a cluster queue.

The format of the name is the same as for the *SecurityExit* parameter described in "Change, Copy, and Create Channel" on page 813.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

ClusterWorkLoadLength **(MQCFIN)**
Cluster workload length (parameter identifier: MQIA_CLUSTER_WORKLOAD_LENGTH).

The maximum length of the message passed to the cluster workload exit.

The value of this attribute must be in the range 0 - 999,999 999.

CLWLMRUChannels **(MQCFIN)**
Cluster workload most recently used (MRU) channels (parameter identifier: MQIA_CLWL_MRU_CHANNELS).

The maximum number of active most recently used outbound channels.

Specify a value in the range 1 - 999,999 999.

CLWLUseQ **(MQCFIN)**
Use of remote queue (parameter identifier: MQIA_CLWL_USEQ).

Specifies whether a cluster queue manager is to use remote puts to other queues defined in other queue managers within the cluster during workload management.

Specify either:

MQCLWL_USEQ_ANY
Use remote queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CodedCharSetId **(MQCFIN)**
Queue manager coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

The coded character set identifier (CCSID) for the queue manager. The CCSID is the identifier used with all character string fields defined by the application programming interface (API). If the CCSID in a message descriptor is set to the value MQCCSI_Q_MGR, it applies to the character data written into the body of a message. Data is written using MQPUT or MQPUT1. Character data is identified by the format specified for the message.

Specify a value in the range 1 - 65,535.

The CCSID must specify a value that is defined for use on the platform and use an appropriate character set. The character set must be:

- EBCDIC on IBM i
- ASCII or ASCII-related on other platforms

Stop and restart the queue manager after execution of this command so that all processes reflect the changed CCSID of the queue manager.

This parameter is not supported on z/OS.

CommandEvent **(MQCFIN)**
Controls whether command events are generated (parameter identifier: MQIA_COMMAND_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_NO_DISPLAY
Event reporting enabled for all successful commands except Inquire commands.

CommandScope **(MQCFIN)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk "*". The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

CommandServerControl **(MQCFIN)**
Specifies whether the command server is to be started when the queue manager starts (parameter identifier: MQIA_CMD_SERVER_CONTROL).

The value can be:

MQSVC_CONTROL_MANUAL
The command server is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The command server is to be started automatically when the queue manager starts.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ConfigurationEvent **(MQCFIN)**
Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Custom **(MQCFST)**
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form NAME(VALUE). Single quotation marks must be escaped with another single quotation mark.

This description is updated when features using this attribute are introduced. Currently there are no possible values for *Custom*.

The maximum length of the string is MQ_CUSTOM_LENGTH.

DeadLetterQName **(MQCFIN)**
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination. The maximum length of the string is MQ_Q_NAME_LENGTH.

DefClusterXmitQueueType **(MQCFIN)**
The DefClusterXmitQueueType attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels. (Parameter identifier: MQIA_DEF_CLUSTER_XMIT_Q_TYPE.)

The values of DefClusterXmitQueueType are MQCLXQ_SCTQ or MQCLXQ_CHANNEL.

MQCLXQ_SCTQ

All cluster-sender channels send messages from SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlID of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute DefClusterXmitQueueType was not present.

MQCLXQ_CHANNEL
Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

The attribute is not supported on z/OS.

DefXmitQName **(MQCFST)**
Default transmission queue name (parameter identifier: MQCA_DEF_XMIT_Q_NAME).

This parameter is the name of the default transmission queue that is used for the transmission of messages to remote queue managers. It is selected if there is no other indication of which transmission queue to use.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DNSGroup **(MQCFST)**
DNS group name (parameter identifier: MQCA_DNS_GROUP).

Specify the name of the group that the TCP listener handling inbound transmissions for the queue-sharing group must join. It must join it when using Workload Manager for Dynamic Domain Name Services support (WLM/DNS). This parameter applies to z/OS only.

The maximum length of the string is MQ_DNS_GROUP_NAME_LENGTH.

DNSWLM **(MQCFIN)**
Controls whether the TCP listener that handles inbound transmissions for the queue-sharing group must register with WLM/DNS: (parameter identifier: MQIA_DNS_WLM).

The value can be:

MQDNSWLM_YES
The listener must register with WLM.

MQDNSWLM_NO
The listener is not to register with WLM. This value is the initial default value of the queue manager.

This parameter applies to z/OS only.

ExpiryInterval **(MQCFIN)**
Interval between scans for expired messages (parameter identifier: MQIA_EXPIRY_INTERVAL). This parameter applies to z/OS only.

Specifies the frequency with which the queue manager scans the queues looking for expired messages. Specify a time interval in seconds in the range 1 - 99,999,999, or the following special value:

MQEXPI_OFF
No scans for expired messages.

The minimum scan interval used is 5 seconds, even if you specify a lower value.

EncryptionPolicySuiteB **(MQCFIL)**
Specifies whether Suite B-compliant cryptography is used and what level of strength is employed (parameter identifier MQIA_SUITE_B_STRENGTH).

The value can be one or more of:

MQ_SUITE_B_NONE
Suite B-compliant cryptography is not used.

MQ_SUITE_B_128_BIT
Suite B 128-bit strength security is used.

MQ_SUITE_B_192_BIT
Suite B 192-bit strength security is used.

If invalid lists are specified, such as MQ_SUITE_B_NONE with MQ_SUITE_B_128_BIT, the error MQRCCF_SUITE_B_ERROR is issued.

Force **(MQCFIN)**
Force changes (parameter identifier: MQIACF_FORCE).

Specifies whether the command is forced to complete if both of the following are true:

- *DefXmitQName* is specified, and
- An application has a remote queue open, the resolution for which is affected by this change.

GroupUR **(MQCFIN)**
Controls whether CICS and XA client applications can establish transactions with a GROUP unit of recovery disposition.

This attribute is only valid on z/OS and can be enabled only when the queue manager is a member of a queue-sharing group.

The value can be:

MQGUR_DISABLED
CICS and XA client applications must connect using a queue manager name.

MQGUR_ENABLED
CICS and XA client applications can establish transactions with a group unit of recovery disposition by specifying a QSG name when they connect.

IGQPutAuthority **(MQCFIN)**
Command scope (parameter identifier: MQIA_IGQ_PUT_AUTHORITY). This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

Specifies the type of authority checking and, therefore, the user IDs to be used by the IGQ agent (IGQA). This parameter establishes the authority to put messages to a destination queue. The value can be:

MQIGQPA_DEFAULT
Default user identifier is used.

The user identifier used for authorization is the value of the *UserIdentifier* field. The *UserIdentifier* field is in the separate MQMD that is associated with the message when the message is on the shared transmission queue. This value is the user identifier of the program that placed the message on the shared transmission queue. It is typically the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) is checked.

MQIGQPA_CONTEXT
Context user identifier is used.

The user identifier used for authorization is the value of the *UserIdentifier* field. The *UserIdentifier* field is in the separate MQMD that is associated with the message when the message is on the shared transmission queue. This value is the user identifier of the program that placed the message on the shared transmission queue. It is typically the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) is checked.. The value of the *UserIdentifier* field in the embedded MQMD is also checked. The latter user identifier is typically the user identifier of the application that originated the message.

MQIGQPA_ONLY_IGQ
Only the IGQ user identifier is used.

The user identifier used for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, this user identifier is used for all checks.

MQIGQPA_ALTERNATE_OR_IGQ
Alternate user identifier or IGQ-agent user identifier is used.

The user identifier used for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the value of the *UserIdentifier* field in the embedded MQMD is also checked. The latter user identifier is typically the user identifier of the application that originated the message.

IGQUserId **(MQCFST)**
Intra-group queuing agent user identifier (parameter identifier: MQCA_IGQ_USER_ID). This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

Specifies the user identifier that is associated with the local intra-group queuing agent. This identifier is one of the user identifiers that might be checked for authorization when the IGQ agent puts messages on local queues. The actual user identifiers checked depend on the setting of the *IGQPutAuthority* attribute, and on external security options.

The maximum length is MQ_USER_ID_LENGTH.

InhibitEvent **(MQCFIN)**
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

IntraGroupQueuing **(MQCFIN)**
Command scope (parameter identifier: MQIA_INTRA_GROUP_QUEUING). This parameter is valid only on z/OS when the queue manager is a member of a queue-sharing group.

Specifies whether intra-group queuing is used. The value can be:

MQIGQ_DISABLED
Intra-group queuing disabled.

MQIGQ_ENABLED
Intra-group queuing enabled.

IPAddressVersion **(MQCFIN)**
IP address version selector (parameter identifier: MQIA_IP_ADDRESS_VERSION).

Specifies which IP address version, either IPv4 or IPv6, is used. The value can be:

MQIPADDR_IPV4
IPv4 is used.

MQIPADDR_IPV6
IPv6 is used.

This parameter is only relevant for systems that run both IPv4 and IPv6. It affects only channels defined as having a *TransportType* of MQXPY_TCP when one of the following conditions is true:

- The channel attribute*ConnectionName* is a host name that resolves to both an IPv4 and IPv6 address and its *LocalAddress* parameter is not specified.
- The channel attributes *ConnectionName* and *LocalAddress* are both host names that resolve to both IPv4 and IPv6 addresses.

ListenerTimer **(MQCFIN)**
Listener restart interval (parameter identifier: MQIA_LISTENER_TIMER).

The time interval, in seconds, between attempts by WebSphere MQ to restart the listener after an APPC or TCP/IP failure. This parameter applies to z/OS only.

Specify a value in the range 5 - 9,999. The initial default value of the queue manager is 60.

LocalEvent **(MQCFIN)**
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

LoggerEvent **(MQCFIN)**
Controls whether recovery log events are generated (parameter identifier: MQIA_LOGGER_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled. This value is valid only on queue managers that use linear logging.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

LUGroupName **(MQCFST)**
Generic LU name for the LU 6.2 listener (parameter identifier: MQCA_LU_GROUP_NAME).

The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group.

This parameter applies to z/OS only.

The maximum length of the string is MQ_LU_NAME_LENGTH.

LUName **(MQCFST)**
LU name to use for outbound LU 6.2 transmissions (parameter identifier: MQCA_LU_NAME).

The name of the LU to use for outbound LU 6.2 transmissions. Set this parameter to be the same as the name of the LU to be used by the listener for inbound transmissions.

This parameter applies to z/OS only.

The maximum length of the string is MQ_LU_NAME_LENGTH.

LU62ARMSuffix **(MQCFST)**
APPCPM suffix (parameter identifier: MQCA_LU62_ARM_SUFFIX).

The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator.

This parameter applies to z/OS only.

The maximum length of the string is MQ_ARM_SUFFIX_LENGTH.

LU62Channels **(MQCFIN)**
Maximum number of LU 6.2 channels (parameter identifier: MQIA_LU62_CHANNELS).

The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

This parameter applies to z/OS only.

Specify a value in the range 0 - 9999. The initial default value of the queue manager is 200.

MaxActiveChannels **(MQCFIN)**
Maximum number of active channels (parameter identifier: MQIA_ACTIVE_CHANNELS).

The maximum number of channels that can be *active* at any time.

This parameter applies to z/OS only.

Sharing conversations do not contribute to the total for this parameter.

Specify a value in the range 1 - 9999. The initial default value of the queue manager is 200.

MaxChannels **(MQCFIN)**
Maximum number of current channels (parameter identifier: MQIA_MAX_CHANNELS).

The maximum number of channels that can be *current* (including server-connection channels with connected clients).

This parameter applies to z/OS only.

Sharing conversations do not contribute to the total for this parameter.

Specify a value in the range 1 - 9999.

MaxHandles **(MQCFIN)**
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

The maximum number of handles that any one connection can have open at the same time.

Specify a value in the range 0 - 999,999,999.

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

Specifies the maximum length of messages allowed on queues on the queue manager. No message that is larger than either the queue attribute *MaxMsgLength* or the queue manager attribute *MaxMsgLength* can be put on a queue.

If you reduce the maximum message length for the queue manager, you must also reduce the maximum message length of the SYSTEM.DEFAULT.LOCAL.QUEUE definition, and your other queues. Reduce the definitions on the queues to less than or equal to the limit of the queue manager. If you do not reduce the message lengths appropriately, and applications inquire only the value of the queue attribute *MaxMsgLength*, they might not work correctly.

The lower limit for this parameter is 32 KB (32,768 bytes). The upper limit is 100 MB (104,857,600 bytes).This parameter is not valid on z/OS.

MaxPropertiesLength **(MQCFIN)**
Maximum property length (parameter identifier: MQIA_MAX_PROPERTIES_LENGTH).

Specifies the maximum length of the properties, including both the property name in bytes and the size of the property value in bytes.

Specify a value in the range 0 - 100 MB (104,857,600 bytes), or the special value:

MQPROP_UNRESTRICTED_LENGTH
The size of the properties is restricted only by the upper limit.

MaxUncommittedMsgs **(MQCFIN)**
Maximum uncommitted messages (parameter identifier: MQIA_MAX_UNCOMMITTED_MSGS).

Specifies the maximum number of uncommitted messages. The maximum number of uncommitted messages under any sync point is the sum of the following messages:

The number of messages that can be retrieved.

The number of messages that can be put.

The number of trigger messages generated within this unit of work.

The limit does not apply to messages that are retrieved or put outside sync point.

Specify a value in the range 1 - 10,000.

MQIAccounting **(MQCFIN)**
Controls whether accounting information for MQI data is to be collected (parameter identifier: MQIA_ACCOUNTING_MQI).

The value can be:

MQMON_OFF
MQI accounting data collection is disabled. This value is the initial default value of the queue manager.

MQMON_ON
MQI accounting data collection is enabled.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

MQIStatistics **(MQCFIN)**
Controls whether statistics monitoring data is to be collected for the queue manager (parameter identifier: MQIA_STATISTICS_MQI).

The value can be:

MQMON_OFF
Data collection for MQI statistics is disabled. This value is the initial default value of the queue manager.

MQMON_ON
Data collection for MQI statistics is enabled.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

MsgMarkBrowseInterval **(MQCFIN)**
Mark-browse interval (parameter identifier: MQIA_MSG_MARK_BROWSE_INTERVAL).

Specifies the time interval in milliseconds after which the queue manager can automatically unmark messages.

Specify a value in the range 0 - 999,999,999, or the special value MQMMBI_UNLIMITED.

A value of 0 causes the queue manager to unmark messages immediately.

MQMMBI_UNLIMITED indicates that the queue manager does not automatically unmark messages.

OutboundPortMax **(MQCFIN)**
The maximum value in the range for the binding of outgoing channels (parameter identifier: MQIA_OUTBOUND_PORT_MAX).

The maximum value in the range of port numbers to be used when binding outgoing channels. This parameter applies to z/OS only.

Specify a value in the range 0 - 65,535. The initial default value of the queue manager is zero.

Specify a corresponding value for *OutboundPortMin* and ensure that the value of *OutboundPortMax* is greater than or equal to the value of *OutboundPortMin*.

OutboundPortMin **(MQCFIN)**
The minimum value in the range for the binding of outgoing channels (parameter identifier: MQIA_OUTBOUND_PORT_MIN).

The minimum value in the range of port numbers to be used when binding outgoing channels. This parameter applies to z/OS only.

Specify a value in the range 0 - 65,535. The initial default value of the queue manager is zero.

Specify a corresponding value for *OutboundPortMax* and ensure that the value of *OutboundPortMin* is less than or equal to the value of *OutboundPortMax*.

Parent **(MQCFST)**
The name of the queue manager to which this queue manager is to connect hierarchically as its child (parameter identifier: MQCA_PARENT).

A blank value indicates that this queue manager has no parent queue manager. If there is an existing parent queue manager it is disconnected. This value is the initial default value of the queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Note:

- The use ofIBM WebSphere MQ hierarchical connections requires that the queue manager attribute PSMode is set to MQPSM_ENABLED.
- The value of *Parent* can be set to a blank value if PSMode is set to MQPSM_DISABLED.
- Before connecting to a queue manager hierarchically as its child, channels in both directions must exist between the parent queue manager and child queue manager.
- If a parent is defined, the **Change Queue Manager** command disconnects from the original parent and sends a connection flow to the new parent queue manager.
- Successful completion of the command does not mean that the action completed or that it is going to complete successfully. Use the **Inquire Pub/Sub Status** command to track the status of the requested parent relationship.

PerformanceEvent **(MQCFIN)**
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

PubSubClus **(MQCFIN)**
Controls whether the queue manager participates in publish/subscribe clustering (parameter identifier: MQIA_PUBSUB_CLUSTER).

The value can be:

MQPSCLUS_ENABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is permitted.

Note: The introduction of a clustered topic into a large IBM WebSphere MQ cluster can cause a degradation in performance. This degradation occurs because all partial repositories are notified of all the other members of the cluster. Unexpected subscriptions might be created at all other nodes; for example; where proxysub(FORCE) is specified. Large numbers of channels might be started from a queue manager; for example, on resync after a queue manager failure.

MQPSCLUS_DISABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is inhibited. The creations or receipts are recorded as warnings in the queue manager error logs.

PubSubMaxMsgRetryCount **(MQCFIN)**
The number of attempts to reprocess a message when processing a failed command message under sync point (parameter identifier: MQIA_PUBSUB_MAXMSG_RETRY_COUNT).

The value can be:

0 to 999 999 999
The initial value is 5.

PubSubMode **(MQCFIN)**
Specifies whether the publish/subscribe engine and the queued publish/subscribe interface are running. The publish/subscribe engine enables applications to publish or subscribe by using the application programming interface. The publish/subscribe interface monitors the queues used the queued publish/subscribe interface (parameter identifier: MQIA_PUBSUB_MODE).

The value can be:

MQPSM_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface. The queued publish/subscribe interface is not running. Any message that is put to the queues that are monitored by the queued publish/subscribe interface are not acted on. Use this setting for compatibility with WebSphere Message Broker V6, or earlier versions. WebSphere Message Broker needs to read the same queues from which the queued publish/subscribe interface normally reads.

MQPSM_DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted on.

MQPSM_ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe by using the application programming interface and the queues that are monitored by the queued publish/subscribe interface. This value is the initial default value of the queue manager.

PubSubNPInputMsg **(MQCFIN)**
Whether to discard (or keep) an undelivered input message (parameter identifier: MQIA_PUBSUB_NP_MSG).

The value can be:

MQUNDELIVERED_DISCARD
Non-persistent input messages are discarded if they cannot be processed.

MQUNDELIVERED_KEEP
Non-persistent input messages are not discarded if they cannot be processed. In this situation, the queued publish/subscribe interface continues to try the process again at appropriate intervals and does not continue processing subsequent messages.

PubSubNPResponse **(MQCFIN)**
Controls the behavior of undelivered response messages (parameter identifier: MQIA_PUBSUB_NP_RESP).

The value can be:

MQUNDELIVERED_NORMAL
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If they cannot be placed on the dead letter queue they are discarded.

MQUNDELIVERED_SAFE
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If the response cannot be sent and cannot be placed on the dead letter queue the queued publish/subscribe interface rolls back the current operation. The operation is tried again at appropriate intervals and does not continue processing subsequent messages.

MQUNDELIVERED_DISCARD
Non-persistent responses that are not placed on the reply queue are discarded.

MQUNDELIVERED_KEEP
Non-persistent responses are not placed on the dead letter queue or discarded. Instead, the queued publish/subscribe interface backs out the current operation and then try it again at appropriate intervals.

PubSubSyncPoint **(MQCFIN)**
Whether only persistent (or all) messages must be processed under sync point (parameter identifier: MQIA_PUBSUB_SYNC_PT).

The value can be:

MQSYNCPOINT_IFPER
This value makes the queued publish/subscribe interface receive non-persistent messages outside sync point. If the interface receives a publication outside sync point, the interface forwards the publication to subscribers known to it outside sync point.

MQSYNCPOINT_YES
This value makes the queued publish/subscribe interface receive all messages under sync point.

QMgrDesc **(MQCFST)**
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

This parameter is text that briefly describes the object.

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager on which the command is executing. Using this character set ensures that the text is translated correctly.

QueueAccounting **(MQCFIN)**
Controls the collection of accounting (thread-level and queue-level accounting) data for queues (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_NONE
Accounting data collection for queues is disabled. This value must not be overridden by the value of the *QueueAccounting* parameter on the queue.

MQMON_OFF
Accounting data collection is disabled for queues specifying a value of MQMON_Q_MGR in the *QueueAccounting* parameter.

MQMON_ON
Accounting data collection is enabled for queues specifying a value of MQMON_Q_MGR in the *QueueAccounting* parameter.

QueueMonitoring **(MQCFIN)**
Default setting for online monitoring for queues (parameter identifier: MQIA_MONITORING_Q).

If the *QueueMonitoring* queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be:

MQMON_OFF
Online monitoring data collection is turned off. This value is the initial default value of the queue manager.

MQMON_NONE
Online monitoring data collection is turned off for queues regardless of the setting of their *QueueMonitoring* attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

QueueStatistics **(MQCFIN)**
Controls whether statistics data is to be collected for queues (parameter identifier: MQIA_STATISTICS_Q).

The value can be:

MQMON_NONE
Statistics data collection is turned off for queues regardless of the setting of their *QueueStatistics* parameter. This value is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection is turned off for queues specifying a value of MQMON_Q_MGR in their *QueueStatistics* parameter.

MQMON_ON
Statistics data collection is turned on for queues specifying a value of MQMON_Q_MGR in their *QueueStatistics* parameter.

This parameter is valid only on IBM i, UNIX, Linux, and Windows systems.

ReceiveTimeout **(MQCFIN)**
How long a TCP/IP channel waits to receive data from its partner (parameter identifier: MQIA_RECEIVE_TIMEOUT).

The approximate length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state.

This parameter applies to z/OS only. It applies to message channels, and not to MQI channels. This number can be qualified as follows:

- This number is a multiplier to be applied to the negotiated *HeartBeatInterval* value to determine how long a channel is to wait. Set *ReceiveTimeoutType* to MQRCVTIME_MULTIPLY. Specify a value of zero or in the range 2 - 99. If you specify zero, the channel waits indefinitely to receive data from its partner.
- This number is a value, in seconds, to be added to the negotiated *HeartBeatInterval* value to determine how long a channel is to wait. Set *ReceiveTimeoutType* to MQRCVTIME_ADD. Specify a value in the range 1 - 999,999.
- Tthis number is a value, in seconds, that the channel is to wait, set *ReceiveTimeoutType* to MQRCVTIME_EQUAL. Specify a value in the range 0 - 999,999. If you specify 0, the channel waits indefinitely to receive data from its partner.

The initial default value of the queue manager is zero.

ReceiveTimeoutMin **(MQCFIN)**
The minimum length of time that a TCP/IP channel waits to receive data from its partner (parameter identifier: MQIA_RECEIVE_TIMEOUT_MIN).

The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. This parameter applies to z/OS only.

Specify a value in the range 0 - 999,999.

ReceiveTimeoutType **(MQCFIN)**
The qualifier to apply to *ReceiveTimeout* (parameter identifier: MQIA_RECEIVE_TIMEOUT_TYPE).

The qualifier to apply to *ReceiveTimeoutType* to calculate how long a TCP/IP channel waits to receive data, including heartbeats, from its partner. It waits to receive data before returning to the inactive state. This parameter applies to z/OS only.

The value can be:

MQRCVTIME_MULTIPLY
The *ReceiveTimeout* value is a multiplier to be applied to the negotiated value of *HeartbeatInterval* to determine how long a channel waits. This value is the initial default value of the queue manager.

MQRCVTIME_ADD
ReceiveTimeout is a value, in seconds, to be added to the negotiated value of *HeartbeatInterval* to determine how long a channel waits.

MQRCVTIME_EQUAL
ReceiveTimeout is a value, in seconds, representing how long a channel waits.

RemoteEvent **(MQCFIN)**
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

RepositoryName **(MQCFST)**
Cluster name (parameter identifier: MQCA_REPOSITORY_NAME).

The name of a cluster for which this queue manager provides a repository manager service.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

No more than one of the resultant values of *RepositoryName* can be nonblank.

RepositoryNamelist **(MQCFST)**
Repository namelist (parameter identifier: MQCA_REPOSITORY_NAMELIST).

The name, of a namelist of clusters, for which this queue manager provides a repository manager service.

This queue manager does not have a full repository, but can be a client of other repository services that are defined in the cluster, if

- Both *RepositoryName* and *RepositoryNamelist* are blank, or
- *RepositoryName* is blank and the namelist specified by *RepositoryNamelist* is empty.

No more than one of the resultant values of *RepositoryNameList* can be nonblank.

SecurityCase **(MQCFIN)**
Security case supported (parameter identifier: MQIA_SECURITY_CASE).

Specifies whether the queue manager supports security profile names in mixed case, or in uppercase only. The value is activated when a Refresh Security command is run with *SecurityType(MQSECTYPE_CLASSES)* specified. This parameter is valid only on z/OS.

The value can be:

MQSCYC_UPPER
Security profile names must be in uppercase.

MQSCYC_MIXED
Security profile names can be in uppercase or in mixed case.

SharedQQmgrName **(MQCFIN)**
Shared-queue queue manager name (parameter identifier: MQIA_SHARED_Q_Q_MGR_NAME).

A queue manager makes an MQOPEN call for a shared queue. The queue manager that is specified in the *ObjectQmgrName* parameter of the MQOPEN call is in the same queue-sharing group as the processing queue manager. The SQQMNAME attribute specifies whether the *ObjectQmgrName* is used or whether the processing queue manager opens the shared queue directly. This parameter is valid only on z/OS.

The value can be:

MQSQQM_USE
ObjectQmgrName is used and the appropriate transmission queue is opened.

MQSQQM_IGNORE
The processing queue manager opens the shared queue directly. This value can reduce the traffic in your queue manager network.

SSLCRLNamelist **(MQCFST)**
The SSL namelist (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The length of the string is MQ_NAMELIST_NAME_LENGTH.

Indicates the name of a namelist of authentication information objects which are used to provide certificate revocation locations to allow enhanced TLS/SSL certificate checking.

If *SSLCRLNamelist* is blank, certificate revocation checking is not invoked.

Changes to *SSLCRLNamelist*, or to the names in a previously specified namelist, or to previously referenced authentication information objects become effective:

- On IBM i, UNIX, Linux, and Windows systems when a new channel process is started.
- For channels that run as threads of the channel initiator on IBM i, UNIX, Linux, and Windows systems, when the channel initiator is restarted.

- For channels that run as threads of the listener on IBM i, UNIX, Linux, and Windows systems, when the listener is restarted.
- On z/OS, when the channel initiator is restarted.
- When a **REFRESH SECURITY TYPE(SSL)** command is issued.
- On IBM i queue managers, this parameter is ignored. However, it is used to determine which authentication information objects are written to the AMQCLCHL.TAB file.

SSLCryptoHardware **(MQCFST)**
The SSL cryptographic hardware (parameter identifier: MQCA_SSL_CRYPTO_HARDWARE).

The length of the string is MQ_SSL_CRYPTO_HARDWARE_LENGTH.

Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

This parameter is supported on UNIX, Linux, and Windows systems only.

All supported cryptographic hardware supports the PKCS #11 interface. Specify a string of the following format:

GSK_PKCS11=*<the PKCS #11 driver path and file name>*;*<the PKCS #11 token label>*;
<the PKCS #11 token password>;*<symmetric cipher setting>*;

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver file name is the name of the shared library. An example of the value required for the PKCS #11 driver path and file name is /usr/lib/pkcs11/PKCS11_API.so

To access symmetric cipher operations through GSKit, specify the symmetric cipher setting parameter. The value of this parameter is either:

SYMMETRIC_CIPHER_OFF
Do not access symmetric cipher operations.

SYMMETRIC_CIPHER_ON
Access symmetric cipher operations.

If the symmetric cipher setting is not specified, this value has the same effect as specifying SYMMETRIC_CIPHER_OFF.

The maximum length of the string is 256 characters. The default value is blank.

If you specify a string in the wrong format, you get an error.

When the SSLCryptoHardware value is changed, the cryptographic hardware parameters specified become the ones used for new SSL connection environments. The new information becomes effective:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.
- When a Refresh Security command is issued to refresh the contents of the SSL key repository.

SSLEvent **(MQCFIN)**
Controls whether SSL events are generated (parameter identifier: MQIA_SSL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

SSLFipsRequired **(MQCFIN)**
SSLFIPS specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in WebSphere MQ, rather than in cryptographic hardware (parameter identifier: MQIA_SSL_FIPS_REQUIRED).

If cryptographic hardware is configured, the cryptographic modules used are those modules provided by the hardware product. These modules might, or might not, be FIPS-certified to a particular level depending on the hardware product in use. This parameter applies to z/OS, UNIX, Linux, and Windows platforms only.

The value can be:

MQSSL_FIPS_NO
WebSphere MQ provides an implementation of SSL cryptography which supplies some FIPS-certified modules on some platforms. If you set *SSLFIPSRequired* to MQSSL_FIPS_NO, any CipherSpec supported on a particular platform can be used. This value is the initial default value of the queue manager.

If the queue manager runs without using cryptographic hardware, refer to the CipherSpecs listed in Specifying CipherSpecs employing FIPS 140-2 certified cryptography:

MQSSL_FIPS_YES
Specifies that only FIPS-certified algorithms are to be used in the CipherSpecs allowed on all SSL connections from and to this queue manager.

For a listing of appropriate FIPS 140-2 certified CipherSpecs; see Specifying CipherSpecs.

Changes to SSLFIPS become effective either:

- On UNIX, Linux, and Windows systems, when a new channel process is started.
- For channels that run as threads of the channel initiator on UNIX, Linux, and Windows systems, when the channel initiator is restarted.
- For channels that run as threads of the listener on UNIX, Linux, and Windows systems, when the listener is restarted.
- For channels that run as threads of a process pooling process, when the process pooling process is started or restarted and first runs an SSL channel. If the process pooling process has already run an SSL channel, and you want the change to become effective immediately, run the MQSC command **REFRESH SECURITY TYPE(SSL)**. The process pooling process is **amqrmppa** on UNIX, Linux, and Windows systems.
- On z/OS, when the channel initiator is restarted.
- When a **REFRESH SECURITY TYPE(SSL)** command is issued, except on z/OS.

SSLKeyRepository **(MQCFST)**
The SSL key repository (parameter identifier: MQCA_SSL_KEY_REPOSITORY).

The length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

Indicates the name of the Secure Sockets Layer key repository.

The format of the name depends on the environment:

- On z/OS, it is the name of a key ring.
- On IBM i, it is of the form *pathname/keyfile*, where *keyfile* is specified without the suffix (.kdb), and identifies a GSKit key database file. The default value is /QIBM/UserData/ICSS/Cert/Server/Default.

  If you specify *SYSTEM, WebSphere MQ uses the system certificate store as the key repository for the queue manager. As a result, the queue manager is registered as a server application in Digital Certificate Manager (DCM). You can assign any server/client certificate in the system store to this application.

If you change the SSLKEYR parameter to a value other than *SYSTEM, WebSphere MQ unregisters the queue manager as an application with DCM.

- On UNIX, it is of the form *pathname*/*keyfile* and on Windows *pathname**keyfile*, where *keyfile* is specified without the suffix (.kdb), and identifies a GSKit key database file. The default value for UNIX platforms is /var/mqm/qmgrs/QMGR/ssl/key, and on Windows it is C:\Program Files\IBM\WebSphere MQ\qmgrs\QMGR\ssl\key, where QMGR is replaced by the queue manager name (on UNIX, Linux, and Windows).

On IBM i, UNIX, Linux, and Windows systems, the syntax of this parameter is validated to ensure that it contains a valid, absolute, directory path.

If SSLKEYR is blank, or is a value that does not correspond to a key ring or key database file, channels using SSL fail to start.

Changes to SSLKeyRepository become effective:

- On IBM i, UNIX, Linux, and Windows platforms, when a new channel process is started.
- For channels that run as threads of the channel initiator on IBM i, UNIX, Linux, and Windows platforms, when the channel initiator is restarted.
- For channels that run as threads of the listener on IBM i, UNIX, Linux, and Windows platforms, when the listener is restarted.
- On z/OS, when the channel initiator is restarted.

SSLKeyResetCount **(MQCFIN)**
SSL key reset count (parameter identifier: MQIA_SSL_RESET_COUNT).

Specifies when SSL channel MCAs that initiate communication reset the secret key used for encryption on the channel. The value of this parameter represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. This number of bytes includes control information sent by the MCA.

The secret key is renegotiated when (whichever occurs first):

- The total number of unencrypted bytes sent and received by the initiating channel MCA exceeds the specified value, or,
- If channel heartbeats are enabled, before data is sent or received following a channel heartbeat.

Specify a value in the range 0 - 999,999,999. A value of zero, the initial default value of the queue manager, signifies that secret keys are never renegotiated. If you specify an SSL/TLS secret key reset count between 1 byte through 32 KB, SSL/TLS channels use a secret key reset count of 32Kb. This count is to avoid the performance effect of excessive key resets which would occur for small SSL/TLS secret key reset values.

SSLTasks **(MQCFIN)**
Number of server subtasks to use for processing SSL calls (parameter identifier: MQIA_SSL_TASKS). This parameter applies to z/OS only.

The number of server subtasks to use for processing SSL calls. To use SSL channels, you must have at least two of these tasks running.

Specify a value in the range 0 - 9999. However, to avoid problems with storage allocation, do not set this parameter to a value greater than 50.

StartStopEvent **(MQCFIN)**
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

StatisticsInterval **(MQCFIN)**
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue (parameter identifier: MQIA_STATISTICS_INTERVAL).

Specify a value in the range 1 - 604,000.

This parameter is valid only on IBM i, UNIX, Linux, and Windows.

TCPChannels **(MQCFIN)**
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol (parameter identifier: MQIA_TCP_CHANNELS).

Specify a value in the range 0 - 9999. The initial default value of the queue manager is 200.

Sharing conversations do not contribute to the total for this parameter.

This parameter applies to z/OS only.

TCPKeepAlive **(MQCFIN)**
Specifies whether the TCP KEEPALIVE facility is to be used to check whether the other end of a connection is still available (parameter identifier: MQIA_TCP_KEEP_ALIVE).

The value can be:

MQTCPKEEP_YES
The TCP KEEPALIVE facility is to be used as specified in the TCP profile configuration data set. The interval is specified in the *KeepAliveInterval* channel attribute.

MQTCPKEEP_NO
The TCP KEEPALIVE facility is not to be used. This value is the initial default value of the queue manager.

This parameter applies to z/OS only.

TCPName **(MQCFST)**
The name of the TCP/IP system that you are using (parameter identifier: MQIA_TCP_NAME).

The maximum length of the string is MQ_TCP_NAME_LENGTH.

This parameter applies to z/OS only.

TCPStackType **(MQCFIN)**
Specifies whether the channel initiator can use only the TCP/IP address space specified in *TCPName*, or can optionally bind to any selected TCP/IP address (parameter identifier: MQIA_TCP_STACK_TYPE).

The value can be:

MQTCPSTACK_SINGLE
The channel initiator uses the TCP/IP address space that is specified in *TCPName*. This value is the initial default value of the queue manager.

MQTCPSTACK_MULTIPLE
The channel initiator can use any TCP/IP address space available to it. It defaults to the one specified in *TCPName* if no other is specified for a channel or listener.

This parameter applies to z/OS only.

TraceRouteRecording **(MQCFIN)**
Specifies whether trace-route information can be recorded and a reply message generated (parameter identifier: MQIA_TRACE_ROUTE_RECORDING).

The value can be:

MQRECORDING_DISABLED
Trace-route information cannot be recorded.

MQRECORDING_MSG
Trace-route information can be recorded and replies sent to the destination specified by the originator of the message causing the trace-route record.

MQRECORDING_Q
Trace-route information can be recorded and replies sent to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

If participation in route tracing is enabled using this queue manager attribute, the value of the attribute is only important if a reply is generated. Route tracing is enabled by not setting *TraceRouteRecording* to MQRECORDING_DISABLED. The reply must go either to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, or to the destination specified by the message itself. Provided the attribute is not disabled then messages not yet at the final destination might have information added to them. For more information about trace-route records, see Controlling trace-route messaging.

TreeLifeTime **(MQCFIN)**
The lifetime, in seconds, of non-administrative topics (parameter identifier: MQIA_TREE_LIFE_TIME).

Non-administrative topics are those topics created when an application publishes to, or subscribes as, a topic string that does not exist as an administrative node. When this non-administrative node no longer has any active subscriptions, this parameter determines how long the queue manager waits before removing that node. Only non-administrative topics that are in use by a durable subscription remain after the queue manager is recycled.

Specify a value in the range 0 - 604,000. A value of 0 means that non-administrative topics are not removed by the queue manager. The initial default value of the queue manager is 1800.

TriggerInterval **(MQCFIN)**
Trigger interval (parameter identifier: MQIA_TRIGGER_INTERVAL).

Specifies the trigger time interval, expressed in milliseconds, for use only with queues where *TriggerType* has a value of MQTT_FIRST.

In this case, trigger messages are normally generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering, even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

Specify a value in the range 0 - 999,999 999.

Error codes (Change Queue Manager)

This command might return the following errors in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHAD_ERROR
Channel automatic definition error.

MQRCCF_CHAD_EVENT_ERROR
Channel automatic definition event error.

MQRCCF_CHAD_EVENT_WRONG_TYPE
Channel automatic definition event parameter not allowed for this channel type.

MQRCCF_CHAD_EXIT_ERROR
Channel automatic definition exit name error.

MQRCCF_CHAD_EXIT_WRONG_TYPE
Channel automatic definition exit parameter not allowed for this channel type.

MQRCCF_CHAD_WRONG_TYPE
Channel automatic definition parameter not allowed for this channel type.

MQRCCF_FORCE_VALUE_ERROR
Force value not valid.

MQRCCF_PATH_NOT_VALID
Path not valid.

MQRCCF_PWD_LENGTH_ERROR
Password length error.

MQRCCF_PSCLUS_DISABLED_TOPDEF
Administrator or application attempted to define a cluster topic when **PubSubClub** is set to MQPSCLUS_DISABLED.

MQRCCF_PSCLUS_TOPIC_EXSITS
Administrator tried to set **PubSubClub** to MQPSCLUS_DISABLED when a cluster topic definition exists.

MQRCCF_Q_MGR_CCSID_ERROR
Coded character set value not valid.

MQRCCF_REPOS_NAME_CONFLICT
Repository names not valid.

MQRCCF_UNKNOWN_Q_MGR
Queue manager not known.

Related information:

Channel states

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

Change, Copy, and Create Service:

The Change Service command changes existing service definitions. The Copy and Create service commands create new service definitions - the Copy command uses attribute values of an existing service definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

The Change Service (MQCMD_CHANGE_SERVICE) command changes the specified attributes of an existing WebSphere MQ service definition. For any optional parameters that are omitted, the value does not change.

The Copy Service (MQCMD_COPY_SERVICE) command creates a WebSphere MQ service definition, using, for attributes not specified in the command, the attribute values of an existing service definition.

The Create Service (MQCMD_CREATE_SERVICE) command creates a WebSphere MQ service definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change and Create Service)

ServiceName **(MQCFST)**
The name of the service definition to be changed or created (parameter identifier: MQCA_SERVICE_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Required parameters (Copy Service)

FromServiceName **(MQCFST)**
The name of the service definition to be copied from (parameter identifier: MQCACF_FROM_SERVICE_NAME).

This parameter specifies the name of the existing service definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ToServiceName **(MQCFST)**
To service name (parameter identifier: MQCACF_TO_SERVICE_NAME).

This parameter specifies the name of the new service definition. If a service definition with this name exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters (Change, Copy, and Create Service)

Replace **(MQCFIN)**
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a namelist definition with the same name as *ToServiceName* exists, this specifies parameter whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

ServiceDesc **(MQCFST)**
Description of service definition (parameter identifier: MQCA_SERVICE_DESC).

This parameter is a plain-text comment that provides descriptive information about the service definition. It must contain only displayable characters.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they might be translated incorrectly.

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceType **(MQCFIN)**
The mode in which the service is to run (parameter identifier: MQIA_SERVICE_TYPE).

Specify either:

MQSVC_TYPE_SERVER
Only one instance of the service can be executed at a time, with the status of the service made available by the Inquire Service Status command.

MQSVC_TYPE_COMMAND
Multiple instances of the service can be started.

StartArguments **(MQCFST)**
Arguments to be passed to the program on startup (parameter identifier: MQCA_SERVICE_START_ARGS).

Specify each argument within the string as you would on a command line, with a space to separate each argument to the program.

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StartCommand **(MQCFST)**
Service program name (parameter identifier: MQCA_SERVICE_START_COMMAND).

Specifies the name of the program which is to run. You must specify a fully qualified path name to the executable program.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

StartMode **(MQCFIN)**
Service mode (parameter identifier: MQIA_SERVICE_CONTROL).

Specifies how the service is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command. This value is the default value.

MQSVC_CONTROL_Q_MGR
The service being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

StderrDestination **(MQCFST)**
Specifies the path to a file to which the standard error (stderr) of the service program must be redirected (parameter identifier: MQCA_STDERR_DESTINATION).

If the file does not exist when the service program is started, the file is created.

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StdoutDestination **(MQCFST)**
Specifies the path to a file to which the standard output (stdout) of the service program must be redirected (parameter identifier: MQCA_STDOUT_DESTINATION).

If the file does not exist when the service program is started, the file is created.

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StopArguments **(MQCFST)**
Specifies the arguments to be passed to the stop program when instructed to stop the service (parameter identifier: MQCA_SERVICE_STOP_ARGS).

Specify each argument within the string as you would on a command line, with a space to separate each argument to the program.

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StopCommand **(MQCFST)**
Service program stop command (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

This parameter is the name of the program that is to run when the service is requested to stop. You must specify a fully qualified path name to the executable program.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

Change, Copy, and Create Subscription:

The Change Subscription command changes existing subscription definitions. The Copy and Create Subscription commands create new subscription definitions - the Copy command uses attribute values of an existing subscription definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

The Change Subscription (MQCMD_CHANGE_SUBSCRIPTION) command changes the specified attributes of an existing WebSphere MQ subscription. For any optional parameters that are omitted, the value does not change.

The Copy Subscription (MQCMD_COPY_SUBSCRIPTION) command creates a WebSphere MQ subscription, using, for attributes not specified in the command, the attribute values of an existing subscription.

The Create Subscription (MQCMD_CREATE_SUBSCRIPTION) command creates a WebSphere MQ administrative subscription so that existing applications can participate in publish/subscribe application.

Required parameters (Change Subscription)

SubName **(MQCFST)**
The name of the subscription definition to be changed (parameter identifier: MQCACF_SUB_NAME).

The maximum length of the string is MQ_SUB_NAME_LENGTH.

or

SubId **(MQCFBS)**
The unique identifier of the subscription definition to be changed (parameter identifier: MQBACF_SUB_ID).

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Required parameters (Copy Subscription)

ToSubscriptionName **(MQCFBS)**
The name of the subscription to copy to (parameter identifier: MQCACF_TO_SUB_NAME).

The maximum length of the string is MQ_SUBSCRIPTION_NAME_LENGTH.

You require at least one of *FromSubscriptionName* or *SubId*.

FromSubscriptionName **(MQCFST)**
The name of the subscription definition to be copied from (parameter identifier: MQCACF_FROM_SUB_NAME).

On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToSubscriptionName* and the disposition MQQSGD_GROUP is used.

The maximum length of the string is MQ_SUBSCRIPTION_NAME_LENGTH.

SubId **(MQCFBS)**
The unique identifier of the subscription definition to be changed (parameter identifier: MQBACF_SUB_ID).

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Required parameters (Create Subscription)

You must provide the *SubName*.

SubName **(MQCFST)**
The name of the subscription definition to be changed (parameter identifier: MQCACF_SUB_NAME).

The maximum length of the string is MQ_SUB_NAME_LENGTH.

You require at least one of *TopicObject* or *TopicString*.

TopicObject **(MQCFST)**
The name of a previously defined topic object from which is obtained the topic name for the subscription (parameter identifier: MQCA_TOPIC_NAME). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString **(MQCFST)**
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING). .

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Optional parameters (Change, Copy, and Create Subscription)

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- a queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Destination **(MQCFST)**
Destination (parameter identifier: MQCACF_DESTINATION).

Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

DestinationClass **(MQCFIN)**
Destination class (parameter identifier: MQIACF_DESTINATION_CLASS).

Specifies whether the destination is managed.

Specify either:

MQDC_MANAGED
The destination is managed.

MQDC_PROVIDED
The destination queue is as specified in the *Destination* field.

Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

DestinationCorrelId **(MQCFBS)**
Destination correlation identifier (parameter identifier: MQBACF_DESTINATION_CORREL_ID).

Provides a correlation identifier that is placed in the *CorrelId* field of the message descriptor for all the messages sent to this subscription.

The maximum length is MQ_CORREL_ID_LENGTH.

DestinationQueueManager **(MQCFST)**
Destination queue manager (parameter identifier: MQCACF_DESTINATION_Q_MGR).

Specifies the name of the destination queue manager, either local or remote, to which messages for the subscription are forwarded.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Expiry **(MQCFIN)**
The time, in tenths of a second, at which a subscription expires after its creation date and time (parameter identifier: MQIACF_EXPIRY).

The default value of unlimited means that the subscription never expires.

After a subscription has expired it becomes eligible to be discarded by the queue manager and receives no further publications.

PublishedAccountingToken **(MQCFBS)**
Value of the accounting token used in the *AccountingToken* field of the message descriptor (parameter identifier: MQBACF_ACCOUNTING_TOKEN).

The maximum length of the string is MQ_ACCOUNTING_TOKEN_LENGTH.

PublishedApplicationIdentifier **(MQCFST)**
Value of the application identity data used in the *ApplIdentityData* field of the message descriptor (parameter identifier: MQCACF_APPL_IDENTITY_DATA.

The maximum length of the string is MQ_APPL_IDENTITY_DATA_LENGTH.

PublishPriority **(MQCFIN)**
The priority of the message sent to this subscription (parameter identifier: MQIACF_PUB_PRIORITY).

The value can be:

MQPRI_PRIORITY_AS_PUBLISHED
Priority of messages sent to this subscription is taken from the priority supplied to the published message. This value is the supplied default value.

MQPRI_PRIORITY_AS_QDEF
Priority of messages sent to this subscription is determined by the default priority of the queue defined as a destination.

0-9 An integer value providing an explicit priority for messages sent to this subscription.

PublishSubscribeProperties **(MQCFIN)**
Specifies how publish/suscribe related message properties are added to messages sent to this subscription (parameter identifier: MQIACF_PUBSUB_PROPERTIES).

The value can be:

MQPSPROP_COMPAT
If the original publication is a PCF message, then the publish/subscribe properties are added as PCF attributes. Otherwise, publish/subscribe properties are added within an MQRFH version 1 header. This method is compatible with applications coded for use with previous versions of WebSphere MQ.

MQPSPROP_NONE
Do not add publish/suscribe properties to the messages. This value is the supplied default value.

MQPSPROP_RFH2
Publish/suscribe properties are added within an MQRFH version 2 header. This method is compatible with applications coded for use with WebSphere Message Brokers.

Selector **(MQCFST)**
Specifies the selector applied to messages published to the topic (parameter identifier: MQCACF_SUB_SELECTOR). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

Only those messages that satisfy the selection criteria are put to the destination specified by this subscription.

The maximum length of the string is MQ_SELECTOR_LENGTH.

SubscriptionLevel **(MQCFIN)**
The level within the subscription interception hierarchy at which this subscription is made (parameter identifier: MQIACF_SUB_LEVEL). To ensure that an intercepting application receives messages before any other subscribers, make sure that it has the highest subscription level of all subscribers.

The value can be:

0 - 9 An integer in the range 0-9. The default value is 1. Subscribers with a subscription level of 9 intercept publications before they reach subscribers with lower subscription levels.

SubscriptionScope **(MQCFIN)**
Determines whether this subscription is passed to other queue managers in the network (parameter identifier: MQIACF_SUBSCRIPTION_SCOPE). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

The value can be:

MQTSCOPE_ALL
The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy. This value is the supplied default value.

MQTSCOPE_QMGR
The subscription only forwards messages published on the topic within this queue manager.

SubscriptionUser **(MQCFST)**
The userid that 'owns' this subscription. This parameter is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription. (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

TopicString **(MQCFST)**
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Userdata **(MQCFST)**
User data (parameter identifier: MQCACF_SUB_USER_DATA).

Specifies the user data associated with the subscription

The maximum length of the string is MQ_USER_DATA_LENGTH.

VariableUser **(MQCFST)**
Specifies whether a user other than the one who created the subscription, that is, the user shown in *SubscriptionUser* can take over the ownership of the subscription (parameter identifier: MQIACF_VARIABLE_USER_ID).

The value can be:

MQVU_ANY_USER
Any user can take over the ownership. This value is the supplied default value.

MQVU_FIXED_USER
No other user can take over the ownership.

WildcardSchema **(MQCFIN)**
Specifies the schema to be used when interpreting any wildcard characters contained in the *TopicString* (parameter identifier: MQIACF_WILDCARD_SCHEMA). Although the parameter is accepted, the value specified cannot be different from the original value for Change Subscription.

The value can be:

MQWS_CHAR
Wildcard characters represent portions of strings for compatibility with WebSphere MQ V6.0 broker.

MQWS_TOPIC
Wildcard characters represent portions of the topic hierarchy for compatibility with WebSphere Message Brokers. This value is the supplied default value.

Change, Copy, and Create Topic:

The Change Topic command changes existing topic definitions. The Copy and Create Topic commands create new topic definitions - the Copy command uses attribute values of an existing topic definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

The Change Topic (MQCMD_CHANGE_TOPIC) command changes the specified attributes of an existing WebSphere MQ administrative topic definition. For any optional parameters that are omitted, the value does not change.

The Copy Topic (MQCMD_COPY_TOPIC) command creates a WebSphere MQ administrative topic definition by using, for attributes not specified in the command, the attribute values of an existing topic definition.

The Create Topic (MQCMD_CREATE_TOPIC) command creates a WebSphere MQ administrative topic definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change Topic)

TopicName **(MQCFST)**
The name of the administrative topic definition to be changed (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Required parameters (Copy Topic)

FromTopicName **(MQCFST)**
The name of the administrative topic object definition to be copied from (parameter identifier: MQCACF_FROM_TOPIC_NAME).

On z/OS, the queue manager searches for an object with the name you specify and a disposition of MQQSGD_Q_MGR or MQQSGD_COPY to copy from. This parameter is ignored if a value of MQQSGD_COPY is specified for *QSGDisposition*. In this case, an object with the name specified by *ToTopicName* and the disposition MQQSGD_GROUP is searched for to copy from.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString **(MQCFST)**
The topic string (parameter identifier: MQCA_TOPIC_STRING). This string uses the forward slash (/) character as a delimiter for elements within the topic tree.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

ToTopicName **(MQCFST)**
The name of the administrative topic definition to copy to (parameter identifier: MQCACF_TO_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Required parameters (Create Topic)

TopicName **(MQCFST)**
The name of the administrative topic definition to be created (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString **(MQCFST)**
The topic string (parameter identifier: MQCA_TOPIC_STRING).

This parameter is required and cannot contain the empty string. The "/" character within this string has a special meaning. It delimits the elements in the topic tree. A topic string can start with the "/" character but is not required to. A string starting with the "/" character is not the same as a string that does not start with the "/" character. A topic string cannot end with the "/" character.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Optional parameters (Change, Copy, and Create Topic)

ClusterName **(MQCFST)**
The name of the cluster to which this topic belongs (parameter identifier: MQCA_CLUSTER_NAME). The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

The value can be:

Blank This topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers.

This value is the default value for this parameter if no value is specified.

String This topic belongs to the indicated cluster.

Additionally, if PublicationScope or SubscriptionScope are set to MQSCOPE_ALL, this value is the cluster to be used for the propagation of publications and subscriptions, for this topic, to publish/subscribe cluster-connected queue managers.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

CommunicationInformation **(MQCFST)**
The Multicast communication information object (parameter identifier: MQCA_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Custom **(MQCFST)**
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form `NAME(VALUE)`. Single quotes must be escaped with another single quote.

This description will be updated when features using this attribute are introduced. At the moment there are no possible values for *Custom*.

DefPersistence **(MQCFIN)**
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE).

Specifies the default for message-persistence of messages published to the topic. Message persistence determines whether messages are preserved across restarts of the queue manager.

The value can be:

MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority **(MQCFIN)**
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Specifies the default priority of messages published to the topic.

Specify either:

integer The default priority to be used, in the range zero through to the maximum priority value that is supported (9).

MQPRI_PRIORITY_AS_PARENT
The default priority is based on the setting of the closest parent administrative topic object in the topic tree.

DefPutResponse **(MQCFIN)**
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The value can be:

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName **(MQCFST)**
Name of the model queue to be used for durable subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions **(MQCFIN)**
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB).

The value can be:

MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.

MQSUB_DURABLE_ALLOWED
Durable subscriptions are permitted.

MQSUB_DURABLE_INHIBITED
Durable subscriptions are not permitted.

InhibitPublications **(MQCFIN)**
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB).

The value can be:

MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_PUB_INHIBITED
Publications are inhibited for this topic.

MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions **(MQCFIN)**
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB).

The value can be:

MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.

MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

Multicast **(MQCFIN)**
Whether multicast is allowable in the topic tree (parameter identifier: MQIA_MULTICAST).

The value can be:

MQMC_AS_PARENT
Whether multicast is allowed on this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQMC_ENABLED
Multicast is allowed on this topic.

MQMC_DISABLED
Multicast is not allowed on this topic.

MQMC_ONLY
Only subscriptions and publications made using multicast are allowed on this topic.

NonDurableModelQName **(MQCFST)**
Name of the model queue to be used for non-durable subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonPersistentMsgDelivery **(MQCFIN)**
The delivery mechanism for non-persistent messages published to this topic (parameter identifier: MQIA_NPM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

PersistentMsgDelivery **(MQCFIN)**
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a

persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ProxySubscriptions **(MQCFIN)**
Whether a proxy subscription is to be sent for this topic to directly connected queue managers, even if no local subscriptions exist (parameter identifier: MQIA_PROXY_SUB).

The value can be:

MQTA_PROXY_SUB_FORCE
A proxy subscription is sent to connected queue managers even if no local subscriptions exist.

Note: The proxy subscription is sent when this value is set on Create or Change of the topic.

MQTA_PROXY_SUB_FIRSTUSE
For each unique topic string at or below this topic object, a proxy subscription is asynchronously sent to all neighboring queue managers in the following scenarios:

- When a local subscription is created.
- When a proxy subscription is received that must be propagated to further directly connected queue managers.

This value is the default value for this parameter if no value is specified.

PublicationScope **(MQCFIN)**
Whether this queue manager propagates publications for this topic, to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE).

The value can be:

MQSCOPE_AS_PARENT
Whether this queue manager propagates publications, for this topic, to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

This value is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Publications for this topic are not propagated to other queue managers.

MQSCOPE_ALL
Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

Note: This behavior can be over-ridden on a publication-by-publication basis, by using MQPMO_SCOPE_QMGR on the Put Message Options.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

QSGDisposition	Change	Copy, Create
MQQSGD_COPY	The object definition resides on the page set of the queue manager that executes the command. The object was defined by using a command that had the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined by using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.	The object is defined on the page set of the queue manager that executes the command by using the MQQSGD_GROUP object of the same name as the *ToTopicName* object (for Copy) or *TopicName* object (for Create).

MQQSGD_GROUP	The object definition resides in the shared repository. The object was defined by using a command that had the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.
	
	If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they refresh local copies on page set zero:
	`DEFINE TOPIC(name)`
	`REPLACE QSGDISP(COPY)`
	
	The Change for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.
		The object definition resides in the shared repository. This definition is allowed only if the queue manager is in a queue-sharing group.
		
		If the definition is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group so that they make or refresh local copies on page set zero:
		`DEFINE TOPIC(name)`
		`REPLACE QSGDISP(COPY)`
		
		The Copy or Create for the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_PRIVATE	The object resides on the page set of the queue manager that executes the command, and was defined with MQQSGD_Q_MGR or MQQSGD_COPY. Any object residing in the shared repository is unaffected.	Not permitted.
MQQSGD_Q_MGR	The object definition resides on the page set of the queue manager that executes the command. The object was defined using a command that had the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command. This value is the default value.	The object is defined on the page set of the queue manager that executes the command. This value is the default value.

Replace **(MQCFIN)**
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a topic definition with the same name as *ToTopicName* exists, this parameter specifies whether it is to be replaced. The value can be as follows:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

SubscriptionScope **(MQCFIN)**
Whether this queue manager propagates subscriptions for this topic, to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE).

The value can be:

MQSCOPE_AS_PARENT
Whether this queue manager propagates subscriptions, for this topic, to queue managers as part of a hierarchy or as part of a publish/subscribe-cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

This value is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Subscriptions for this topic are not propagated to other queue managers.

MQSCOPE_ALL
Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

Note: This behavior can be over-ridden on a subscription-by-subcription basis, by using MQSO_SCOPE_QMGR on the Subscription Descriptor or SUBSCOPE(QMGR) on DEFINE SUB.

TopicDesc **(MQCFST)**
Topic description (parameter identifier: MQCA_TOPIC_DESC).

Text that briefly describes the object

The maximum length is MQ_TOPIC_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing to ensure that the text is translated correctly if it is sent to another queue manager.

TopicType **(MQCFIN)**
Topic type (parameter identifier: MQIA_TOPIC_TYPE).

The value specified must match the type of the topic being changed. The value can be:

MQTOPT_LOCAL
Local topic object

UseDLQ **(MQCFIN)**
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value can be:

MQUSEDLQ_AS_PARENT
Determines whether to use the dead-letter queue using the setting of the closest administrative topic object in the topic tree. This value is the default supplied with WebSphere MQ, but your installation might have changed it.

MQUSEDLQ_NO
Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message. The MQPUT of an application to a topic fails in accordance with the settings of MQIA_NPM_DELIVERY and MQIA_PM_DELIVERY.

MQUSEDLQ_YES
If the DEADQ queue manager attribute provides the name of a dead-letter queue then it is used, otherwise the behavior is as for MQUSEDLQ_NO.

WildcardOperation **(MQCFIN)**
Behavior of subscriptions including wildcards made to this topic (parameter identifier: MQIA_WILDCARD_OPERATION).

The value can be:

MQTA_PASSTHRU
A less specific wildcard subscription is a subscription made by using wildcard topic names that are less specific than the topic string at this topic object. MQTA_PASSTHRU lets less specific wildcard subscriptions receive publications made to this topic and to topic strings more specific than this topic. This value is the default supplied with WebSphere MQ.

MQTA_BLOCK
A less specific wildcard subscription is a subscription made by using wildcard topic names that are less specific than the topic string at this topic object. MQTA_BLOCK stops less specific wildcard subscriptions receiving publications made to this topic or to topic strings more specific than this topic.

This value of this attribute is used when subscriptions are defined. If you alter this attribute, the set of topics covered by existing subscriptions is not affected by the modification. This value applies also, if the topology is changed when topic objects are created or deleted; the set of topics matching subscriptions created following the modification of the *WildcardOperation* attribute is created by using the modified topology. If you want to force the matching set of topics to be re-evaluated for existing subscriptions, you must restart the queue manager.

Clear Queue:

The Clear Queue (MQCMD_CLEAR_Q) command deletes all the messages from a local queue.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

The command fails if the queue contains uncommitted messages.

Required parameters

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the local queue to be cleared. The maximum length of the string is MQ_Q_NAME_LENGTH.

Note: The target queue must be type local.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

MQQSGD_PRIVATE
Clear the private queue named in *QName*. The queue is private if it was created using a command with the attributes MQQSGD_PRIVATE or MQQSGD_Q_MGR. This value is the default value.

MQQSGD_SHARED
Clear the shared queue named in *QName*. The queue is shared if it was created using a command with the attribute MQQSGD_SHARED. This value applies only to local queues.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

This reason occurs only if there are uncommitted updates.

MQRCCF_Q_WRONG_TYPE
Action not valid for the queue of specified type.

Clear Topic String:

The Clear Topic String (MQCMD_CLEAR_TOPIC_STRING) command clears the retained message which is stored for the specified topic.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

TopicString **(MQCFST)**
Topic String (parameter identifier: MQCA_TOPIC_STRING).

The topic string to be cleared The maximum length of the string is MQ_TOPIC_STR_LENGTH.

ClearType **(MQCFIN)**
Clear type (parameter identifier: MQIACF_CLEAR_TYPE).

Specifies the type of clear command being issued. The value must be:

MQCLRT_RETAINED Remove the retained publication from the specified topic string.

Optional parameters

Scope **(MQCFIN)**
Scope of clearance (parameter identifier: MQIACF_CLEAR_SCOPE).

Whether the topic string is to be cleared locally or globally. The value can be:

MQCLRS_LOCAL
The retained message is removed from the specified topic string at the local queue manager only.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Delete Authentication Information Object:

The Delete authentication information (MQCMD_DELETE_AUTH_INFO) command deletes the specified authentication information object.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

AuthInfoName **(MQCFST)**
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

MQQSGD_COPY
The object definition resides on the page set of the queue manager which executes this command. The object was defined by a command using the parameter MQQSGD_COPY. Any object in the shared repository, or any object defined by a command using the parameter MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE AUTHINFO(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Delete Authority Record:

The Delete Authority Record (MQCMD_DELETE_AUTH_REC) command deletes an authority record. The authorizations associated with the profile no longer apply to WebSphere MQ objects with names that match the profile name specified.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ObjectType **(MQCFIN)**
The type of object for which to delete authorizations (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

ProfileName **(MQCFST)**
Name of the profile to be deleted (parameter identifier: MQCACF_AUTH_PROFILE_NAME).

If you have defined a generic profile then you can specify it here, using wildcard characters to specify a named generic profile to be removed. If you specify an explicit profile name, the object must exist.

The maximum length of the string is MQ_AUTH_PROFILE_NAME_LENGTH.

Optional parameters

GroupNames **(MQCFSL)**
Group names (parameter identifier: MQCACF_GROUP_ENTITY_NAMES).

The names of groups having a profile deleted. At least one group name or principal name must be specified. An error occurs if neither are specified.

Each member in this list can be a maximum length of MQ_ENTITY_NAME_LENGTH.

PrincipalNames **(MQCFSL)**
Principal names (parameter identifier: MQCACF_PRINCIPAL_ENTITY_NAMES).

The names of principals having a profile deleted. At least one group name or principal name must be specified. An error occurs if neither are specified.

Each member in this list can be a maximum length of MQ_ENTITY_NAME_LENGTH.

Error codes (Delete Authority Record)

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRC_OBJECT_TYPE_ERROR
Invalid object type.

MQRC_UNKNOWN_ENTITY
Userid not authorized, or unknown.

MQRCCF_ENTITY_NAME_MISSING
Entity name missing.

MQRCCF_OBJECT_TYPE_MISSING
Object type missing.

MQRCCF_PROFILE_NAME_ERROR
Invalid profile name.

Delete Channel:

The Delete Channel (MQCMD_DELETE_CHANNEL) command deletes the specified channel definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel definition to be deleted. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

None of the following attributes are applicable to MQTT channels unless specifically mentioned in the parameter description.

ChannelType **(MQCFIN)**
The type of channel (parameter identifier: MQIACH_CHANNEL_TYPE). This parameter is currently only used with MQTT Telemetry channels, and is required when deleting a Telemetry channel. The only value that can currently be given to the parameter is **MQCHT_MQTT**.

ChannelTable **(MQCFIN)**
Channel table (parameter identifier: MQIACH_CHANNEL_TABLE).

Specifies the ownership of the channel definition table that contains the specified channel definition.

The value can be:

MQCHTAB_Q_MGR
Queue-manager table.

MQCHTAB_Q_MGR is the default. This table contains channel definitions for channels of all types except MQCHT_CLNTCONN.

MQCHTAB_CLNTCONN
Client-connection table.

This table only contains channel definitions for channels of type MQCHT_CLNTCONN.

This parameter is not applicable to IBM WebSphere MQ Telemetry.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined by a command using the parameter MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameters MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE CHANNEL(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 802.

Error codes

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TABLE_ERROR
Channel table value not valid.

Delete Channel (MQTT):

The Delete Telemetry Channel (MQCMD_DELETE_CHANNEL) command deletes the specified channel definition.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel definition to be deleted. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType **(MQCFIN)**
The type of channel (parameter identifier: MQIACH_CHANNEL_TYPE). Required when deleting a Telemetry channel. The only value that can currently be given to the parameter is **MQCHT_MQTT**.

Optional parameters

None of the following attributes are applicable to MQTT channels unless specifically mentioned in the parameter description.

ChannelTable **(MQCFIN)**
Channel table (parameter identifier: MQIACH_CHANNEL_TABLE).

Specifies the ownership of the channel definition table that contains the specified channel definition.

The value can be:

MQCHTAB_Q_MGR
Queue-manager table.

MQCHTAB_Q_MGR is the default. This table contains channel definitions for channels of all types except MQCHT_CLNTCONN.

MQCHTAB_CLNTCONN
Client-connection table.

This table only contains channel definitions for channels of type MQCHT_CLNTCONN.

This parameter is not applicable to IBM WebSphere MQ Telemetry.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined by a command using the parameter MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameters MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE CHANNEL(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 802.

Error codes

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TABLE_ERROR
Channel table value not valid.

Delete Channel Listener:

The Delete Channel Listener (MQCMD_DELETE_LISTENER) command deletes an existing channel listener definition.

HP Integrity NonStop Server	UNIX and Linux systems	Windows
	X	X

Required parameters

ListenerName **(MQCFST)**
Listener name (parameter identifier: MQCACH_LISTENER_NAME).

This parameter is the name of the listener definition to be deleted. The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Delete Communication Information Object:

The Delete Communication Information Object (MQCMD_DELETE_COMM_INFO) command deletes the specified communication information object.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameter

ComminfoName **(MQCFST)**
The name of the communication information definition to be deleted (parameter identifier: MQCA_COMM_INFO_NAME).

Delete Namelist:

The Delete Namelist (MQCMD_DELETE_NAMELIST) command deletes an existing namelist definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

NamelistName **(MQCFST)**
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

This parameter is the name of the namelist definition to be deleted. The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any

object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE NAMELIST(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Delete Process:

The Delete Process (MQCMD_DELETE_PROCESS) command deletes an existing process definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

ProcessName **(MQCFST)**
Process name (parameter identifier: MQCA_PROCESS_NAME).

The process definition to be deleted. The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the command is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE PROCESS(name) QSGDISP(COPY)
```

The deletion of the group object takes effect regardless of whether the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Delete Queue:

The Delete Queue (MQCMD_DELETE_Q) command deletes a queue.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be deleted.

If the *Scope* attribute of the queue is MQSCO_CELL, the entry for the queue is deleted from the cell directory.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

Authrec **(MQCFIN)**
Authrec (parameter identifier: MQIACF_REMOVE_AUTHREC).

Specifies whether the associated authority record is also deleted.

This parameter does not apply to z/OS.

The value can be:

MQRAR_YES
The authority record associated with the object is deleted. This is the default.

MQRAR_NO
The authority record associated with the object is not deleted.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Purge **(MQCFIN)**
Purge queue (parameter identifier: MQIACF_PURGE).

If there are messages on the queue MQPO_YES must be specified, otherwise the command fails. If this parameter is not present the queue is not purged.

Valid only for queue of type local.

The value can be:

MQPO_YES
Purge the queue.

MQPO_NO
Do not purge the queue.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the deletion is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to delete local copies on page set zero:

```
DELETE queue(q-name) QSGDISP(COPY)
```

or, for a local queue only:

```
DELETE QLOCAL(q-name) NOPURGE QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

Note: You always get the NOPURGE option even if you specify MQPO_YES for *Purge*. To delete messages on local copies of the queues, you must explicitly issue, for each copy, the Delete Queue command with a *QSGDisposition* value of MQQSGD_COPY and a *Purge* value of MQPO_YES.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

MQQSGD_SHARED
Valid only for queue of type local.

The object resides in the shared repository. The object was defined by a command using the parameter MQQSGD_SHARED. Any object residing on the page set of the queue manager that executes the command, or any object defined by a command using the parameter MQQSGD_GROUP, is not affected by this command.

QType **(MQCFIN)**
Queue type (parameter identifier: MQIA_Q_TYPE).

If this parameter is present, the queue must be of the specified type.

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

Error codes (Delete Queue)

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 802.

Reason **(MQLONG)**
The value can be:

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

Delete Service:

The Delete Service (MQCMD_DELETE_SERVICE) command deletes an existing service definition.

HP Integrity NonStop Server	UNIX and Linux systems	Windows
	X	X

Required parameters

ServiceName **(MQCFST)**
Service name (parameter identifier: MQCA_SERVICE_NAME).

This parameter is the name of the service definition to be deleted.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Delete Subscription:

The Delete Subscription (MQCMD_DELETE_SUBSCRIPTION) command deletes a subscription.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

SubName **(MQCFST)**
Subscription name (parameter identifier: MQCACF_SUB_NAME).

Specifies the unique subscription name. The subscription name, if provided, must be fully specified; a wildcard is not acceptable.

The subscription name must refer to a durable subscription.

If *SubName* is not provided, *SubId* must be specified to identify the subscription to be deleted.

The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId **(MQCFBS)**
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

Specifies the unique internal subscription identifier.

You must supply a value for *SubId* if you have not supplied a value for *SubName*.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- A queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

- An asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter on which to filter.

Delete Topic:

The Delete Topic (MQCMD_DELETE_TOPIC) command deletes the specified administrative topic object.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

TopicName **(MQCFST)**
The name of the administrative topic definition to be deleted (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Optional parameters

Authrec **(MQCFIN)**
Authrec (parameter identifier: MQIACF_REMOVE_AUTHREC).

Specifies whether the associated authority record is also deleted.

This parameter does not apply to z/OS.

The value can be:

MQRAR_YES
The authority record associated with the object is deleted. This is the default.

MQRAR_NO
The authority record associated with the object is not deleted.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object to which you are applying the command (that is, where it is defined and how it behaves). The value can be:

MQQSGD_COPY
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_COPY. Any object residing in the shared repository, or any object defined using a command that had the parameters MQQSGD_Q_MGR, is not affected by this command.

MQQSGD_GROUP
The object definition resides in the shared repository. The object was defined by a command using the parameter MQQSGD_GROUP. Any object residing on the page set of the queue manager that executes the command (except a local copy of the object) is not affected by this command.

If the deletion is successful, the following MQSC command is generated and sent to all active queue managers in the queue-sharing group to make, or delete, local copies on page set zero:

```
DELETE TOPIC(name) QSGDISP(COPY)
```

The deletion of the group object takes effect even if the generated command with QSGDISP(COPY) fails.

MQQSGD_Q_MGR
The object definition resides on the page set of the queue manager that executes the command. The object was defined by a command using the parameter MQQSGD_Q_MGR. Any object residing in the shared repository, or any local copy of such an object, is not affected by this command.

MQQSGD_Q_MGR is the default value.

Escape:

The Escape (MQCMD_ESCAPE) command conveys any WebSphere MQ command (MQSC) to a remote queue manager.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Use the Escape command when the queue manager (or application) sending the command does not support the particular WebSphere MQ command, and so does not recognize it and cannot construct the required PCF command.

The Escape command can also be used to send a command for which no Programmable Command Format has been defined.

The only type of command that can be carried is one that is identified as an MQSC, that is recognized at the receiving queue manager.

Required parameters

EscapeType **(MQCFIN)**
Escape type (parameter identifier: MQIACF_ESCAPE_TYPE).

The only value supported is:

MQET_MQSC
WebSphere MQ command.

EscapeText **(MQCFST)**
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string to hold a command. The length of the string is limited only by the size of the message.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_ESCAPE_TYPE_ERROR
Escape type not valid.

Escape (Response):

The response to the Escape (MQCMD_ESCAPE) command consists of the response header followed by two parameter structures, one containing the escape type, and the other containing the text response. More than one such message might be issued, depending upon the command contained in the Escape request.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

The *Command* field in the response header MQCFH contains the MQCMD_* command identifier of the text command contained in the *EscapeText* parameter in the original Escape command. For example, if *EscapeText* in the original Escape command specified `PING QMGR`, *Command* in the response has the value MQCMD_PING_Q_MGR.

If it is possible to determine the outcome of the command, the *CompCode* in the response header identifies whether the command was successful. The success or otherwise can therefore be determined without the recipient of the response having to parse the text of the response.

If it is not possible to determine the outcome of the command, *CompCode* in the response header has the value MQCC_UNKNOWN, and *Reason* is MQRC_NONE.

Parameters

EscapeType **(MQCFIN)**
Escape type (parameter identifier: MQIACF_ESCAPE_TYPE).

The only value supported is:

MQET_MQSC
WebSphere MQ command.

EscapeText **(MQCFST)**
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string holding the response to the original command.

Inquire Authentication Information Object:

The Inquire authentication information object (MQCMD_INQUIRE_AUTH_INFO) command inquires about the attributes of authentication information objects.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

AuthInfoName **(MQCFST)**
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

Specifies the name of the authentication information object about which information is to be returned.

Generic authentication information object names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all authentication information objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

Optional parameters

AuthInfoAttrs (MQCFIL)
Authentication information object attributes (parameter identifier: MQIACF_AUTH_INFO_ATTRS).

The attribute list can specify the following value - the default value if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
Date on which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCA_AUTH_INFO_DESC
Description of the authentication information object.

MQCA_AUTH_INFO_NAME
Name of the authentication information object.

MQIA_AUTH_INFO_TYPE
Type of authentication information object.

MQCA_AUTH_INFO_CONN_NAME
Connection name of the authentication information object.

MQCA_LDAP_USER_NAME
LDAP user name in the authentication information object.

MQCA_LDAP_PASSWORD
LDAP password in the authentication information object.

MQCA_AUTH_INFO_OCSP_URL
The URL of the OCSP responder used to check for certificate revocation.

AuthInfoType (MQCFIN)
Type of authentication information object. The following values are accepted:

MQAIT_CRL_LDAP
Authentication information objects specifying Certificate Revocation Lists held on LDAP servers.

MQAIT_OCSP
Authentication information objects specifying certificate revocation checking using OCSP.

MQAIT_ALL
Authentication information objects of any type.

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand (MQCFIF)
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *AuthInfoAttrs*, except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. This value is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. This value is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand (MQCFSF)
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *AuthInfoAttrs*, except MQCA_AUTH_INFO_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Inquire Authentication Information Object (Response):

The response of the Inquire authentication information (MQCMD_INQUIRE_AUTH_INFO) command consists of the response header followed by the *AuthInfoName* structure (and on z/OS only, the *QSGDisposition* structure), and the requested combination of attribute parameter structures (where applicable).

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Always returned:
AuthInfoName, *QSGDisposition*

Returned if requested:
AlterationDate, *AlterationTime*, *AuthInfoConnName*, *AuthInfoDesc*, *AuthInfoType*, *LDAPPassword*, *LDAPUserName*

Response data

AlterationDate (MQCFST)
Alteration date of the authentication information object, in the form yyyy-mm-dd (parameter identifier: MQCA_ALTERATION_DATE).

AlterationTime (MQCFST)
Alteration time of the authentication information object, in the form hh.mm.ss (parameter identifier: MQCA_ALTERATION_TIME).

AuthInfoConnName (MQCFST)
The connection name of the authentication information object (parameter identifier: MQCA_AUTH_INFO_CONN_NAME).

The maximum length of the string is MQ_AUTH_INFO_CONN_NAME_LENGTH. On z/OS, it is MQ_LOCAL_ADDRESS_LENGTH.

AuthInfoDesc (MQCFST)
The description of the authentication information object (parameter identifier: MQCA_AUTH_INFO_DESC).

The maximum length is MQ_AUTH_INFO_DESC_LENGTH.

AuthInfoName (MQCFST)
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value can be:

MQAIT_CRL_LDAP
This authentication information object specifies Certificate Revocation Lists that are held on LDAP servers.

MQAIT_OCSP
This authentication information object specifies certificate revocation checking using OCSP.

See for more information.

LDAPPassword (MQCFST)
The LDAP password (parameter identifier: MQCA_LDAP_PASSWORD).

The maximum length is MQ_LDAP_PASSWORD_LENGTH.

LDAPUserName (MQCFST)
The LDAP user name (parameter identifier: MQCA_LDAP_USER_NAME).

The Distinguished Name of the user who is binding to the directory.

The maximum length is MQ_DISTINGUISHED_NAME_LENGTH. On z/OS, it is MQ_SHORT_DNAME_LENGTH.

OCSPResponderURL (MQCFST)
The URL of the OCSP responder used to check for certificate revocation.

QSGDisposition (MQCFIN)
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Authentication Information Object Names:

The Inquire authentication information names (MQCMD_INQUIRE_AUTH_INFO_NAMES) command asks for a list of authentication information names that match the generic authentication information name specified.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

AuthInfoName **(MQCFST)**
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

Specifies the name of the authentication information object about which information is to be returned.

Generic authentication information object names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all authentication information objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

Optional parameters

AuthInfoType (MQCFIN)
Type of authentication information object. The following values are accepted:

MQAIT_CRL_LDAP
Authentication information objects specifying Certificate Revocation Lists held on LDAP servers.

MQAIT_OCSP
Authentication information objects specifying certificate revocation checking using OCSP.

MQAIT_ALL
Authentication information objects of any type. MQAIT_ALL is the default value

CommandScope (MQCFST)
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition (MQCFIN)
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Authentication Information Object Names (Response):

The response to the inquire authentication information names (MQCMD_INQUIRE_AUTH_INFO_NAMES) command consists of the response header followed by a parameter structure giving zero or more names that match the specified authentication information name.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Additionally, on z/OS only, a parameter structure, *QSGDispositions* (with the same number of entries as the *AuthInfoNames* structure), is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *AuthInfoNames* structure.

Always returned:
AuthInfoNames, *QSGDispositions*

Returned if requested:
None

Response data

AuthInfoNames **(MQCFSL)**
List of authentication information object names (parameter identifier: MQCACF_AUTH_INFO_NAMES).

QSGDispositions **(MQCFIL)**
List of QSG dispositions (parameter identifier: MQIACF_QSG_DISPS).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Authority Records:

The Inquire Authority Records (MQCMD_INQUIRE_AUTH_RECS) command retrieves authority records associated with a profile name.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

Options **(MQCFIN)**
Options to control the set of authority records that is returned (parameter identifier: MQIACF_AUTH_OPTIONS).

This parameter is required and you must include one of the following two values:

MQAUTHOPT_NAME_ALL_MATCHING
Return all profiles the names of which match the specified *ProfileName*. This means that a *ProfileName* of ABCD results in the profiles ABCD, ABC*, and AB* being returned (if ABC* and AB* have been defined as profiles).

MQAUTHOPT_NAME_EXPLICIT
Return only those profiles the names of which exactly match the *ProfileName*. No matching generic profiles are returned unless the *ProfileName* is, itself, a generic profile. You cannot specify this value and MQAUTHOPT_ENTITY_SET.

and one of the following two values:

MQAUTHOPT_ENTITY_EXPLICIT
Return all profiles the entity fields of which match the specified *EntityName*. No profiles are returned for any group in which *EntityName* is a member; only the profile defined for the specified *EntityName*.

MQAUTHOPT_ENTITY_SET
Return the profile the entity field of which matches the specified *EntityName* and the profiles pertaining to any groups in which *EntityName* is a member that contribute to the cumulative authority for the specified entity. You cannot specify this value and MQAUTHOPT_NAME_EXPLICIT.

You can also optionally specify:

MQAUTHOPT_NAME_AS_WILDCARD
Interpret *ProfileName* as a filter on the profile name of the authority records. If you do not specify this attribute and *ProfileName* contains wildcard characters, it is interpreted as a generic profile and only those authority records where the generic profile names match the value of *ProfileName* are returned.

You cannot specify MQAUTHOPT_NAME_AS_WILDCARD if you also specify MQAUTHOPT_ENTITY_SET.

ProfileName **(MQCFST)**
Profile name (parameter identifier: MQCACF_AUTH_PROFILE_NAME).

This parameter is the name of the profile for which to retrieve authorizations. Generic profile names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all profiles having names that start with the selected character string. An asterisk on its own matches all possible names.

If you have defined a generic profile, you can return information about it by not setting MQAUTHOPT_NAME_AS_WILDCARD in *Options*.

If you set *Options* to MQAUTHOPT_NAME_AS_WILDCARD, the only valid value for *ProfileName* is a single asterisk (*). This means that all authority records that satisfy the values specified in the other parameters are returned.

Do not specify *ProfileName* if the value of *ObjectType* is MQOT_Q_MGR.

The profile name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_AUTH_PROFILE_NAME_LENGTH.

ObjectType **(MQCFIN)**
The type of object referred to by the profile (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:

MQOT_ALL
All object types. MQOT_ALL is the default if you do not specify a value for *ObjectType*.

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

Optional parameters

EntityName **(MQCFST)**
Entity name (parameter identifier: MQCACF_ENTITY_NAME).

Depending on the value of *EntityType*, this parameter is either:

- A principal name. This name is the name of a user for whom to retrieve authorizations to the specified object. On WebSphere MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.
- A group name. This name is the name of the user group on which to make the inquiry. You can specify one name only and this name must be the name of an existing user group.

For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

```
GroupName@domain
domain\GroupName
```

The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

EntityType **(MQCFIN)**
Entity type (parameter identifier: MQIACF_ENTITY_TYPE).

The value can be:

MQZAET_GROUP
The value of the *EntityName* parameter refers to a group name.

MQZAET_PRINCIPAL
The value of the *EntityName* parameter refers to a principal name.

ProfileAttrs **(MQCFIL)**
Profile attributes (parameter identifier: MQIACF_AUTH_PROFILE_ATTRS).

The attribute list might specify the following value on its own - the default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCACF_ENTITY_NAME
Entity name.

MQIACF_AUTHORIZATION_LIST
Authorization list.

MQIACF_ENTITY_TYPE
Entity type.

Note: If an entity is specified by using the parameters MQCACF_ENTITY_NAME and MQIACF_ENTITY_TYPE, then all the required parameters must be passed in first, in the following order:

1. MQIACF_AUTH_OPTIONS
2. MQIACF_OBJECT_TYPE
3. MQIACF_ENTITY_TYPE
4. MQCACF_ENTITY_NAME

ServiceComponent **(MQCFST)**
Service component (parameter identifier: MQCACF_SERVICE_COMPONENT).

If installable authorization services are supported, this parameter specifies the name of the authorization service from which to retrieve authorization.

If you omit this parameter, the authorization inquiry is made to the first installable component for the service.

The maximum length of the string is MQ_SERVICE_COMPONENT_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRC_OBJECT_TYPE_ERROR
Invalid object type.

MQRC_UNKNOWN_ENTITY
User ID not authorized, or unknown.

MQRCCF_CFST_CONFLICTING_PARM
Conflicting parameters.

MQRCCF_PROFILE_NAME_ERROR
Invalid profile name.

MQRCCF_ENTITY_NAME_MISSING
Entity name missing.

MQRCCF_OBJECT_TYPE_MISSING
Object type missing.

MQRCCF_PROFILE_NAME_MISSING
Profile name missing.

Inquire Authority Records (Response):

The response to the Inquire Authority Records (MQCMD_INQUIRE_AUTH_RECS) command consists of the response header followed by the *QMgrName*, *Options*, *ProfileName*, and *ObjectType* structures and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

One PCF message is returned for each authority record that is found the profile name of which matches the options specified in the Inquire Authority Records request.

Always returned:
ObjectType, *Options*, *ProfileName*, *QMgrName*

Returned if requested:
AuthorizationList, *EntityName*, *EntityType*

Response data

AuthorizationList **(MQCFIL)**
Authorization list (parameter identifier: MQIACF_AUTHORIZATION_LIST).

This list can contain zero or more authorization values. Each returned authorization value means that any user ID in the specified group or principal has the authority to perform the operation defined by that value. The value can be:

MQAUTH_NONE
The entity has authority set to 'none'.

MQAUTH_ALT_USER_AUTHORITY
Specify an alternate user ID on an MQI call.

MQAUTH_BROWSE
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

MQAUTH_CHANGE
Change the attributes of the specified object, using the appropriate command set.

MQAUTH_CLEAR
Clear a queue.

MQAUTH_CONNECT
Connect the application to the specified queue manager by issuing an MQCONN call.

MQAUTH_CREATE
Create objects of the specified type using the appropriate command set.

MQAUTH_DELETE
Delete the specified object using the appropriate command set.

MQAUTH_DISPLAY
Display the attributes of the specified object using the appropriate command set.

MQAUTH_INPUT
Retrieve a message from a queue by issuing an MQGET call.

MQAUTH_INQUIRE
Make an inquiry on a specific queue by issuing an MQINQ call.

MQAUTH_OUTPUT
Put a message on a specific queue by issuing an MQPUT call.

MQAUTH_PASS_ALL_CONTEXT
Pass all context.

MQAUTH_PASS_IDENTITY_CONTEXT
Pass the identity context.

MQAUTH_SET
Set attributes on a queue from the MQI by issuing an MQSET call.

MQAUTH_SET_ALL_CONTEXT
Set all context on a queue.

MQAUTH_SET_IDENTITY_CONTEXT
Set the identity context on a queue.

MQAUTH_CONTROL
For listeners and services, start and stop the specified channel, listener, or service.

For channels, start, stop, and ping the specified channel.

For topics, define, alter, or delete subscriptions.

MQAUTH_CONTROL_EXTENDED
Reset or resolve the specified channel.

MQAUTH_PUBLISH
Publish to the specified topic.

MQAUTH_SUBSCRIBE
Subscribe to the specified topic.

MQAUTH_RESUME
Resume a subscription to the specified topic.

MQAUTH_SYSTEM
Use queue manager for internal system operations.

MQAUTH_ALL
Use all operations applicable to the object.

MQAUTH_ALL_ADMIN
Use all operations applicable to the object.

MQAUTH_ALL_MQI
Use all MQI calls applicable to the object.

Use the *Count* field in the MQCFIL structure to determine how many values are returned.

EntityName **(MQCFST)**
Entity name (parameter identifier: MQCACF_ENTITY_NAME).

This parameter can either be a principal name or a group name.

The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

EntityType **(MQCFIN)**
Entity type (parameter identifier: MQIACF_ENTITY_TYPE).

The value can be:

MQZAET_GROUP
The value of the *EntityName* parameter refers to a group name.

MQZAET_PRINCIPAL
The value of the *EntityName* parameter refers to a principal name.

MQZAET_UNKNOWN
On Windows, an authority record still exists from a previous queue manager which did not originally contain entity type information.

ObjectType **(MQCFIN)**
Object type (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

Options **(MQCFIN)**
Options used to indicate the level of information that is returned (parameter identifier: MQIACF_AUTH_OPTIONS).

ProfileName **(MQCFST)**
Profile name (parameter identifier: MQCACF_AUTH_PROFILE_NAME).

The maximum length of the string is MQ_AUTH_PROFILE_NAME_LENGTH.

QMgrName **(MQCFST)**
Name of the queue manager on which the Inquire command is issued (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Inquire Authority Service:

The Inquire Authority Service (MQCMD_INQUIRE_AUTH_SERVICE) command retrieves information about the level of function supported by installed authority managers.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

AuthServiceAttrs **(MQCFIL)**
Authority service attributes (parameter identifier: MQIACF_AUTH_SERVICE_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQIACF_INTERFACE_VERSION
Current interface version of the authority service.

MQIACF_USER_ID_SUPPORT
Whether the authority service supports user IDs.

Optional parameters

ServiceComponent **(MQCFST)**
Name of authorization service (parameter identifier: MQCACF_SERVICE_COMPONENT).

The name of the authorization service which is to handle the Inquire Authority Service command.

If this parameter is omitted, or specified as a blank or null string, the inquire function is called in each installed authorization service in reverse order to the order in which the services have been installed, until all authorization services have been called or until one returns a value of MQZCI_STOP in the Continuation field.

The maximum length of the string is MQ_SERVICE_COMPONENT_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 802.

Reason **(MQLONG)**
The value can be:

MQRC_SELECTOR_ERROR
Attribute selector not valid.

MQRC_UNKNOWN_COMPONENT_NAME
Unknown service component name.

Inquire Authority Service (Response):

The response to the Inquire Authority Service (MQCMD_INQUIRE_AUTH_SERVICE) command consists of the response header followed by the *ServiceComponent* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned:
ServiceComponent

Returned if requested:
InterfaceVersion, *UserIDSupport*

Response data

InterfaceVersion **(MQCFIN)**
Interface version (parameter identifier: MQIACF_INTERFACE_VERSION).

This parameter is the current interface version of the OAM.

ServiceComponent **(MQCFSL)**
Name of authorization service (parameter identifier: MQCACF_SERVICE_COMPONENT).

If you included a specific value for *ServiceComponent* on the Inquire Authority Service command, this field contains the name of the authorization service that handled the command. If you did not include a specific value for *ServiceComponent* on the Inquire Authority Service command, the list contains the names of all the installed authorization services.

If there is no OAM or if the OAM requested in the ServiceComponent does not exist this field is blank.

The maximum length of each element in the list is MQ_SERVICE_COMPONENT_LENGTH.

UserIDSupport **(MQCFIN)**
User ID support (parameter identifier: MQIACF_USER_ID_SUPPORT).

The value can be:

MQUIDSUPP_YES
The authority service supports user IDs.

MQUIDSUPP_NO
The authority service does not support user IDs.

Inquire Channel:

The Inquire Channel (MQCMD_INQUIRE_CHANNEL) command inquires about the attributes of IBM WebSphere MQ channel definitions.

HP Integrity NonStop Server	UNIX and Linux	Windows
✓	✓	✓

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all channels having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelAttrs **(MQCFIL)**
Channel attributes (parameter identifier: MQIACF_CHANNEL_ATTRS).

The attribute list can specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the parameters in the following table:

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
MQCA_ALTERATION_DATE Date on which the definition was last altered	✓	✓	✓	✓	✓	✓	✓	✓
MQCA_ALTERATION_TIME Time at which the definition was last altered	✓	✓	✓	✓	✓	✓	✓	✓
MQCA_CLUSTER_NAME Name of local queue manager							✓	✓
MQCA_CLUSTER_NAMELIST Name of local queue manager							✓	✓
MQCA_Q_MGR_NAME Name of local queue manager					✓			
MQCACH_CHANNEL_NAME Channel name. You cannot use this attribute as a filter keyword.	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_CONNECTION_NAME Connection name	✓	✓		✓	✓		✓	✓

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
MQCACH_DESC Description	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_LOCAL_ADDRESS Local communications address for the channel	✓	✓		✓	✓		✓	✓
MQCACH_MCA_NAME Message channel agent name	✓	✓		✓			✓	
MQCACH_MCA_USER_ID MCA user identifier	✓	✓	✓	✓		✓	✓	✓
MQCACH_MODE_NAME Mode name	✓	✓		✓	✓		✓	✓
MQCACH_MR_EXIT_NAME Message-retry exit name			✓	✓				✓
MQCACH_MR_EXIT_USER_DATA Message-retry exit name			✓	✓				✓
MQCACH_MSG_EXIT_NAME Message exit name	✓	✓	✓	✓			✓	✓
MQCACH_MSG_EXIT_USER_DATA Message exit user data	✓	✓	✓	✓			✓	✓
MQCACH_PASSWORD Password	✓	✓		✓	✓		✓	
MQCACH_RCV_EXIT_NAME Receive exit name	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_RCV_EXIT_USER_DATA Receive exit user data	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_SEC_EXIT_NAME Security exit name	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_SEC_EXIT_USER_DATA Security exit user data	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_SEND_EXIT_NAME Send exit name	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_SEND_EXIT_USER_DATA Send exit user data	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_SSL_CIPHER_SPEC SSL cipher spec	✓	✓	✓	✓	✓	✓	✓	✓

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
MQCACH_SSL_PEER_NAME SSL peer name	✓	✓	✓	✓	✓	✓	✓	✓
MQCACH_TP_NAME Transaction program name	✓	✓		✓	✓	✓	✓	✓
MQCACH_USER_ID User identifier	✓	✓		✓	✓		✓	
MQCACH_XMIT_Q_NAME Transmission queue name	✓	✓						
MQIA_MONITORING_CHANNEL Online monitoring data collection	✓	✓	✓	✓		✓	✓	✓
MQIA_PROPERTY_CONTROL Property control attribute	✓	✓					✓	✓
MQIA_STATISTICS_CHANNEL Online statistics collection	✓	✓	✓	✓			✓	✓
MQIA_USE_DEAD_LETTER_Q Determines whether the dead-letter queue is used when messages cannot be delivered by channels.	✓	✓	✓	✓			✓	✓
MQIACH_BATCH_HB Value to use for batch heartbeating	✓	✓					✓	✓
MQIACH_BATCH_INTERVAL Batch wait interval (seconds)	✓	✓					✓	✓
MQIACH_BATCH_DATA_LIMIT Batch data limit (kilobytes)	✓	✓					✓	✓
MQIACH_BATCH_SIZE Batch size	✓	✓	✓	✓			✓	✓
MQIACH_CHANNEL_TYPE Channel type	✓	✓	✓	✓	✓	✓	✓	✓
MQIACH_CLIENT_CHANNEL_WEIGHT Client Channel Weight					✓			
MQIACH_CLWL_CHANNEL_ PRIORITY Cluster workload channel priority							✓	✓
MQIACH_CLWL_CHANNEL_RANK Cluster workload channel rank							✓	✓
MQIACH_CLWL_CHANNEL_WEIGHT Cluster workload channel weight							✓	✓

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
MQIACH_CONNECTION_ AFFINITY Connection Affinity					✓			
MQIACH_DATA_CONVERSION Whether sender must convert application data	✓	✓					✓	✓
MQIACH_DEF_RECONNECT Default reconnection option					✓			
MQIACH_DISC_INTERVAL Disconnection interval	✓	✓				✓	✓	✓
MQIACH_HB_INTERVAL Heartbeat interval (seconds)	✓	✓	✓	✓	✓	✓	✓	✓
MQIACH_HDR_COMPRESSION List of header data compression techniques supported by the channel	✓	✓	✓	✓	✓	✓	✓	✓
MQIACH_KEEP_ALIVE_INTERVAL KeepAlive interval	✓	✓	✓	✓	✓	✓	✓	✓
MQIACH_LONG_RETRY Long retry count	✓	✓					✓	✓
MQIACH_LONG_TIMER Long timer	✓	✓					✓	✓
MQIACH_MAX_INSTANCES Maximum number of simultaneous instances of a server-connection channel that can be started.						✓		
MQIACH_MAX_INSTS_PER_ CLIENT Maximum number of simultaneous instances of a server-connection channel that can be started from a single client.						✓		
MQIACH_MAX_MSG_LENGTH Maximum message length	✓	✓	✓	✓	✓	✓	✓	✓
MQIACH_MCA_TYPE MCA type	✓	✓		✓			✓	✓
MQIACH_MR_COUNT Message retry count			✓	✓				✓
MQIACH_MSG_COMPRESSION List of message data compression techniques supported by the channel	✓	✓	✓	✓	✓	✓	✓	✓

Parameter	Sender	Server	Receiver	Requester	Client conn	Server conn	Cluster sender	Cluster receiver
MQIACH_MR_INTERVAL Message retry interval (milliseconds)			✓	✓				✓
MQIACH_NPM_SPEED Speed of nonpersistent messages	✓	✓	✓	✓			✓	✓
MQIACH_PUT_AUTHORITY Put authority			✓	✓		✓		✓
MQIACH_RESET_REQUESTED Sequence number of outstanding request when a RESET CHANNEL command is used	✓	✓	✓	✓			✓	✓
MQIACH_SEQUENCE_NUMBER_ WRAP Sequence number wrap	✓	✓	✓	✓			✓	✓
MQIACH_SHARING_CONVERSATIONS Value of Sharing Conversations						✓		
MQIACH_SHORT_RETRY Short retry count	✓	✓					✓	✓
MQIACH_SHORT_TIMER Short timer	✓	✓					✓	✓
MQIACH_SSL_CLIENT_AUTH SSL client authentication	✓	✓	✓	✓		✓		✓
MQIACH_XMIT_PROTOCOL_TYPE Transport (transmission protocol) type	✓	✓	✓	✓	✓	✓	✓	✓

Note:

1. Only one of the following parameters can be specified:
   - MQCACH_JAAS_CONFIG
   - MQCACH_MCA_USER_ID
   - MQIACH_USE_CLIENT_ID

If none of these parameters are specified, no authentication is performed. If MQCACH_JAAS_CONFIG is specified, the client flows a user name and password, in all other cases the flowed user name is ignored.

ChannelType **(MQCFIN)**

Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If this parameter is present, eligible channels are limited to the specified type. Any attribute selector specified in the *ChannelAttrs* list which is only valid for channels of a different type or types is ignored; no error is raised.

If this parameter is not present (or if MQCHT_ALL is specified), channels of all types other than MQCHT_MQTT are eligible. Each attribute specified must be a valid channel attribute selector (that is, it must be one from the following list), but it might not be applicable to all (or any) of the channels returned. Channel attribute selectors that are valid but not applicable to the channel are ignored, no error messages occur, and no attribute is returned.

The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

MQCHT_MQTT
Telemetry channel.

MQCHT_ALL
All types other than MQCHT_MQTT.

The default value if this parameter is not specified is MQCHT_ALL.

Note: If this parameter is present, it must occur immediately after the *ChannelName* parameter on platforms other than z/OS otherwise resulting in a MQRCCF_MSG_LENGTH_ERROR error message.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

DefaultChannelDisposition **(MQCFIN)**
Default channel disposition (parameter identifier: MQIACH_CHANNEL_DISP).

This parameter is not allowed for client-connection (CLNTCONN) channels.

This parameter applies to z/OS only.

Specifies the disposition of the channels for which information is to be returned. If this parameter is not present (or if MQCHLD_ALL is specified), channels of all channel dispositions are eligible. The value can be:

MQCHLD_ALL
Returns requested information for all eligible channels.

MQCHLD_PRIVATE
Returns requested information for PRIVATE channels.

MQCHLD_SHARED
Returns requested information for channels with channel disposition that is defined as either MQCHLD_SHARED or MQCHLD_FIXSHARED.

DefReconnect **(MQCFIN)**
Client channel default reconnection option (parameter identifier: MQIACH_DEF_RECONNECT).

The default automatic client reconnection option. You can configure a IBM WebSphere MQ MQI client to automatically reconnect a client application. The IBM WebSphere MQ MQI client tries to reconnect to a queue manager after a connection failure. It tries to reconnect without the application client issuing an MQCONN or MQCONNX MQI call.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ChannelAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter for channel type, you cannot also specify the *ChannelType* parameter.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ChannelAttrs* except MQCACH_CHANNEL_NAME and MQCACH_MCA_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

Inquire Channel (MQTT):

The Inquire Channel (MQCMD_INQUIRE_CHANNEL) command inquires about the attributes of IBM WebSphere MQ channel definitions.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all channels having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If this parameter is present, eligible channels are limited to the specified type. Any attribute selector specified in the *ChannelAttrs* list which is only valid for channels of a different type or types is ignored; no error is raised.

If this parameter is not present (or if MQCHT_ALL is specified), channels of all types are eligible. Each attribute specified must be a valid channel attribute selector (that is, it must be one from the following list), but it might not be applicable to all (or any) of the channels returned. Channel attribute selectors that are valid but not applicable to the channel are ignored, no error messages occur, and no attribute is returned.

The value must be:

MQCHT_MQTT
Telemetry channel.

Optional parameters

ChannelAttrs **(MQCFIL)**
Channel attributes (parameter identifier: MQIACF_CHANNEL_ATTRS).

The attribute list can specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following parameters:

MQCA_SSL_KEY_REPOSITORY
SSL Key Repository

MQCACH_CHANNEL_NAME
Channel name. You cannot use this attribute as a filter keyword.

MQCACH_JAAS_CONFIG
The file path of the JAAS configuration

MQCACH_LOCAL_ADDRESS
Local communications address for the channel

MQCACH_MCA_USER_ID
MCA user identifier.

MQCACH_SSL_CIPHER_SPEC
SSL cipher spec.

MQCACH_SSL_KEY_PASSPHRASE
SSL key passphrase.

MQIACH_BACKLOG
The number of concurrent connection requests that the channel supports.

MQIACH_CHANNEL_TYPE
Channel type

MQIACH_PORT
Port number to use when *TransportType* is set to TCP.

MQIACH_SSL_CLIENT_AUTH
SSL client authentication.

MQIACH_USE_CLIENT_ID
Specify whether to use the *clientID* of a new connection as the *userID* for that connection

MQIACH_XMIT_PROTOCOL_TYPE
Transport (transmission protocol) type

Note:

1. Only one of the following parameters can be specified:
   - MQCACH_JAAS_CONFIG
   - MQCACH_MCA_USER_ID
   - MQIACH_USE_CLIENT_ID

If none of these parameters are specified, no authentication is performed. If MQCACH_JAAS_CONFIG is specified, the client flows a user name and password, in all other cases the flowed user name is ignored.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

Inquire Channel (Response):

The response to the Inquire Channel (MQCMD_INQUIRE_CHANNEL) command consists of the response header followed by the *ChannelName* and *ChannelType* structures (and on z/OS only, the *DefaultChannelDisposition*, and *QSGDisposition* structure), and the requested combination of attribute parameter structures (where applicable).

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

If a generic channel name was specified, one such message is generated for each channel found.

Always returned:
ChannelName, ChannelType, DefaultChannelDisposition, QSGDisposition

Returned if requested:
AlterationDate, AlterationTime, BatchHeartbeat, BatchInterval , BatchSize , ChannelDesc, ChannelMonitoring, ChannelStartTime, ChannelStartDate, ChannelStatistics , ClientChannelWeight , ClientIdentifier , ClusterName, ClusterNamelist, CLWLChannelPriority, CLWLChannelRank, CLWLChannelWeight , ConnectionAffinity , ConnectionName , DataConversion, DefReconnect, DiscInterval, HeaderCompression , HeartbeatInterval , InDoubtInbound , InDoubtOutbound, KeepAliveInterval, LastMsgTime, LocalAddress , LongRetryCount , LongRetryInterval , MaxMsgLength, MCAName, MCAType , MCAUserIdentifier , MessageCompression , ModeName, MsgExit, MsgRetryCount , MsgRetryExit , MsgRetryInterval , MsgRetryUserData, MsgsReceived, MsgsSent, MsgUserData , NetworkPriority , NonPersistentMsgSpeed , Password, PendingOutbound, PropertyControl, PutAuthority, QMgrName , ReceiveExit , ReceiveUserData, ResetSeq, SecurityExit, SecurityUserData , SendExit , SendUserData, SeqNumberWrap, SharingConversations, ShortRetryCount, ShortRetryInterval, SSLCipherSpec , SSLCipherSuite , SSLClientAuth , SSLPeerName, TpName, TransportType , UseDLQ , UserIdentifier, XmitQName

Response data

AlterationDate **(MQCFST)**
Alteration date, in the form `yyyy-mm-dd` (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time, in the form `hh.mm.ss` (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

BatchHeartbeat **(MQCFIN)**
The value being used for the batch heartbeating (parameter identifier: MQIACH_BATCH_HB).

The value can be 0 - 999999. A value of 0 indicates that heartbeating is not in use.

BatchInterval **(MQCFIN)**
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

BatchSize **(MQCFIN)**
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

ChannelDesc **(MQCFST)**
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

ChannelMonitoring **(MQCFIN)**
Online monitoring data collection (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_OFF
Online monitoring data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's *ChannelMonitoring* parameter is inherited by the channel.

MQMON_LOW
Online monitoring data collection is turned on, with a low rate of data collection, for this channel unless the queue manager's *ChannelMonitoring* parameter is MQMON_NONE.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate rate of data collection, for this channel unless the queue manager's *ChannelMonitoring* parameter is MQMON_NONE.

MQMON_HIGH
Online monitoring data collection is turned on, with a high rate of data collection, for this channel unless the queue manager's *ChannelMonitoring* parameter is MQMON_NONE.

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStartDate **(MQCFST)**
The date that the channel started (parameter identifier: MQCACH_CHANNEL_START_DATE). The length is specified by the value MQ_DATE_LENGTH.

ChannelStartTime **(MQCFST)**
The time that the channel started (parameter identifier: MQCACH_CHANNEL_START_TIME). The length is specified by the value MQ_TIME_LENGTH.

ChannelStatistics **(MQCFIN)**
Statistics data collection (parameter identifier: MQIA_STATISTICS_CHANNEL).

The value can be:

MQMON_OFF
Statistics data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's *ChannelStatistics* parameter is inherited by the channel.

MQMON_LOW
Statistics data collection is turned on, with a low rate of data collection, for this channel unless the queue manager's *ChannelStatistics* parameter is MQMON_NONE.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate rate of data collection, for this channel unless the queue manager's *ChannelStatistics* parameter is MQMON_NONE.

MQMON_HIGH
Statistics data collection is turned on, with a high rate of data collection, for this channel unless the queue manager's *ChannelStatistics* parameter is MQMON_NONE.

This parameter is valid only on Windows, UNIX and Linux systems.

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

MQCHT_MQTT
Telemetry channel.

ClientChannelWeight **(MQCFIN)**
Client Channel Weight (parameter identifier: MQIACH_CLIENT_CHANNEL_WEIGHT).

The client channel weighting attribute is used so client channel definitions can be selected at random, with the larger weightings having a higher probability of selection, when more than one suitable definition is available.

The value can be 0 - 99. The default is 0.

This parameter is only valid for channels with a ChannelType of MQCHT_CLNTCONN

ClientIdentifier **(MQCFST)**
the clientId of the client (parameter identifier: MQCACH_CLIENT_ID).

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterNamelist **(MQCFST)**
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

CLWLChannelPriority **(MQCFIN)**
Channel priority (parameter identifier: MQIACH_CLWL_CHANNEL_PRIORITY).

CLWLChannelRank **(MQCFIN)**
Channel rank (parameter identifier: MQIACH_CLWL_CHANNEL_RANK).

CLWLChannelWeight **(MQCFIN)**
Channel weighting (parameter identifier: MQIACH_CLWL_CHANNEL_WEIGHT).

ConnectionAffinity **(MQCFIN)**
Channel Affinity (parameter identifier: MQIACH_CONNECTION_AFFINITY)

The channel affinity attribute specifies whether client applications that connect multiple times using the same queue manager name, use the same client channel. The value can be:

MQCAFTY_PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any zero ClientChannelWeight definitions first in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful nonzero ClientChannelWeight definitions are moved to the end of the list. Zero ClientChannelWeight definitions remain at the start of the list and are selected first for each connection. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created. Each client process with the same host name creates the same list.

MQCAFTY_PREFERRED is the default value.

MQCAFTY_NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process independently select an applicable definition based on the weighting with any applicable zero ClientChannelWeight definitions selected first in alphabetical order. For C, C++ and .NET (including fully managed .NET) clients the list is updated if the CCDT has been modified since the list was created.

This parameter is only valid for channels with a ChannelType of MQCHT_CLNTCONN.

ConnectionName **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH. On z/OS, it is MQ_LOCAL_ADDRESS_LENGTH.

The *ConnectionName* is a comma-separated list.

DataConversion **(MQCFIN)**
Whether sender must convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DefaultChannelDisposition **(MQCFIN)**
Default channel disposition (parameter identifier: MQIACH_DEF_CHANNEL_DISP).

This parameter applies to z/OS only.

Specifies the intended disposition of the channel when active. The value can be:

MQCHLD_PRIVATE
The intended use of the object is as a private channel.

MQCHLD_FIXSHARED
The intended use of the object is as a shared channel linked to a specific queue manager.

MQCHLD_SHARED
The intended use of the object is as a shared channel.

DiscInterval **(MQCFIN)**
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

DefReconnect **(MQCFIN)**
Client channel default reconnection option (parameter identifier: MQIACH_DEF_RECONNECT).

The returned values can be:

MQRCN_NO
MQRCN_NO is the default value.

Unless overridden by MQCONNX, the client is not reconnected automatically.

MQRCN_YES
Unless overridden by MQCONNX, the client reconnects automatically.

MQRCN_Q_MGR
Unless overridden by MQCONNX, the client reconnects automatically, but only to the same queue manager. The QMGR option has the same effect as MQCNO_RECONNECT_Q_MGR.

MQRCN_DISABLED
Reconnection is disabled, even if requested by the client program using the MQCONNX MQI call.

HeaderCompression **(MQCFIL)**
Header data compression techniques supported by the channel (parameter identifier: MQIACH_HDR_COMPRESSION). For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval **(MQCFIN)**
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

InDoubtInbound **(MQCFIN)**
Number of inbound messages to the client that are in doubt (Parameter identifier: MQIACH_IN_DOUBT_IN).

InDoubtOutbound **(MQCFIN)**
Number of outbound messages from the client that are in doubt (Parameter identifier: MQIACH_IN_DOUBT_OUT).

KeepAliveInterval **(MQCFIN)**
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

LastMsgTime **(MQCFST)**
The time that the last message was sent or received (parameter identifier: MQCACH_LAST_MSG_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

LocalAddress **(MQCFST)**
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetryCount **(MQCFIN)**
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

LongRetryInterval **(MQCFIN)**
Long timer (parameter identifier: MQIACH_LONG_TIMER).

MaxInstances **(MQCFIN)**
Maximum number of simultaneous instances of a server-connection channel (parameter identifier: MQIACH_MAX_INSTANCES).

This parameter is returned only for server-connection channels in response to an Inquire Channel call with ChannelAttrs including MQIACF_ALL or MQIACH_MAX_INSTANCES.

MaxInstancesPerClient **(MQCFIN)**
Maximum number of simultaneous instances of a server-connection channel that can be started from a single client (parameter identifier: MQIACH_MAX_INSTS_PER_CLIENT).

This parameter is returned only for server-connection channels in response to an Inquire Channel call with ChannelAttrs including MQIACF_ALL or MQIACH_MAX_INSTS_PER_CLIENT.

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

MCAName **(MQCFST)**
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

The maximum length of the string is MQ_MCA_NAME_LENGTH.

MCAType **(MQCFIN)**
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

The value can be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (Windows only).

MCAUserIdentifier **(MQCFST)**
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

Note: An alternative way of providing a user ID for a channel to run under is to use channel authentication records. With channel authentication records, different connections can use the same channel while using different credentials. If both MCAUSER on the channel is set and channel authentication records are used to apply to the same channel, the channel authentication records take precedence. The MCAUSER on the channel definition is only used if the channel authentication record uses USERSRC(CHANNEL). For more details, see Channel authentication records

The maximum length of the MCA user identifier depends on the environment in which the MCA is running. MQ_MCA_USER_ID_LENGTH gives the maximum length for the environment for which your application is running. MQ_MAX_MCA_USER_ID_LENGTH gives the maximum for all supported environments.

On Windows, the user identifier might be qualified with the domain name in the following format:

```
user@domain
```

MessageCompression **(MQCFIL)**
Message data compression techniques supported by the channel (parameter identifier: MQIACH_MSG_COMPRESSION). For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of:

MQCOMPRESS_NONE
No message data compression is performed.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_ANY
Any compression technique supported by the queue manager can be used. MQCOMPRESS_ANY is only valid for receiver, requester, and server-connection channels.

ModeName **(MQCFST)**
Mode name (parameter identifier: MQCACH_MODE_NAME).

The maximum length of the string is MQ_MODE_NAME_LENGTH.

MsgExit **(MQCFST)**
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

In the following environments, if more than one message exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure: IBM i, Windows, UNIX and Linux. An MQCFSL structure is always used on z/OS.

MsgsReceived **(MQCFIN64)**
The number of messages received by the client since it last connected (parameter identifier: MQIACH_MSGS_RECEIVED / MQIACH_MSGS_RCVD).

MsgRetryCount **(MQCFIN)**
Message retry count (parameter identifier: MQIACH_MR_COUNT).

MsgRetryExit **(MQCFST)**
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

MsgRetryInterval **(MQCFIN)**
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

MsgRetryUserData **(MQCFST)**
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgsSent **(MQCFIN64)**
The number of messages sent by the client since it last connected (parameter identifier: MQIACH_MSGS_SENT).

MsgUserData **(MQCFST)**
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, if more than one message exit user data string has been defined for the channel, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure: IBM i, Windows, UNIX and Linux. An MQCFSL structure is always used on z/OS.

NetworkPriority **(MQCFIN)**
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

NonPersistentMsgSpeed **(MQCFIN)**
Speed at which non-persistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password **(MQCFST)**
Password (parameter identifier: MQCACH_PASSWORD).

If a nonblank password is defined, it is returned as asterisks. Otherwise, it is returned as blanks.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

PropertyControl **(MQCFIN)**
Property control attribute (parameter identifier MQIA_PROPERTY_CONTROL).

Specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor). The value can be:

MQPROP_COMPATIBILITY

Message properties	Result
The message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All optional message properties (where the **Support** value is MQPD_SUPPORT_OPTIONAL), except those properties in the message descriptor or extension, are placed in one or more MQRFH2 headers in the message data before the message it sent to the remote queue manager.
The message does not contain a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**	All message properties, except those properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.
The message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL	The message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.
The message contains one or more properties where the **Support** field of the property descriptor is set to MQPD_SUPPORT_OPTIONAL but other fields of the property descriptor are set to non-default values	The properties with non-default values are removed from the message before the message is sent to the remote queue manager.
The MQRFH2 folder that would contain the message property needs to be assigned with the *content='properties'* attribute	The properties are removed to prevent MQRFH2 headers with unsupported syntax flowing to a V6 or prior queue manager.

MQPROP_NONE
All properties of the message, except those properties in the message descriptor or extension, are removed from the message before the message is sent to the remote queue manager.

If the message contains a property where the **Support** field of the property descriptor is not set to MQPD_SUPPORT_OPTIONAL then the message is rejected with reason MQRC_UNSUPPORTED_PROPERTY and treated in accordance with its report options.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

This attribute is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

PutAuthority **(MQCFIN)**
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

The value can be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

QMgrName **(MQCFST)**
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid only on z/OS. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

ReceiveExit **(MQCFST)**
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

In the following environments, if more than one receive exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure: IBM i, Windows, UNIX and Linux. An MQCFSL structure is always used on z/OS.

ReceiveUserData **(MQCFST)**
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, if more than one receive exit user data string has been defined for the channel, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure: IBM i, Windows, UNIX and Linux. An MQCFSL structure is always used on z/OS.

ResetSeq **(MQCFIN)**

Pending reset sequence number.

This is the sequence number from an outstanding request and it indicates a user Reset Channel command request is outstanding.

A value of zero indicates that there is no outstanding Reset Channel. The value can be in the range 1 - 999999999.

Possible return values include MQCHRR_RESET_NOT_REQUESTED.

This parameter is not applicable on z/OS.

SecurityExit **(MQCFST)**
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

SecurityUserData **(MQCFST)**
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit **(MQCFST)**
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

In the following environments, if more than one send exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure: IBM i, Windows, UNIX and Linux. An MQCFSL structure is always used on z/OS.

SendUserData **(MQCFST)**
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, if more than one send exit user data string has been defined for the channel, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure: IBM i, Windows, UNIX and Linux. An MQCFSL structure is always used on z/OS.

SeqNumberWrap **(MQCFIN)**
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

SharingConversations **(MQCFIN)**
Number of sharing conversations (parameter identifier: MQIACH_SHARING_CONVERSATIONS).

This parameter is returned only for TCP/IP client-connection and server-connection channels.

ShortRetryCount **(MQCFIN)**
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

ShortRetryInterval **(MQCFIN)**
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

SSLCipherSpec **(MQCFST)**
CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

SSLCipherSuite **(MQCFST)**
CipherSuite (parameter identifier: MQCACH_SSL_CIPHER_SUITE).

The length of the string is MQ_SSL_CIPHER_SUITE_LENGTH.

SSLClientAuth **(MQCFIN)**
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be

MQSCA_REQUIRED
Client authentication required

MQSCA_OPTIONAL
Client authentication is optional.

Defines whether IBM WebSphere MQ requires a certificate from the SSL client.

SSLPeerName **(MQCFST)**
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

Note: An alternative way of restricting connections into channels by matching against the SSL or TLS Subject Distinguished Name, is to use channel authentication records. With channel authentication records, different SSL or TLS Subject Distinguished Name patterns can be applied to the same channel. If both SSLPEER on the channel and a channel authentication record are used to apply to the same channel, the inbound certificate must match both patterns in order to connect. For more information, see Channel authentication records .

The length of the string is MQ_SSL_PEER_NAME_LENGTH. On z/OS, it is MQ_SSL_SHORT_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the SSL certificate.) If the Distinguished Name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

TpName **(MQCFST)**
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The maximum length of the string is MQ_TP_NAME_LENGTH.

TransportType **(MQCFIN)**
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value might be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

MQXPT_DECNET
DECnet.

UseDLQ **(MQCFIN)**
Whether the dead-letter queue (or undelivered message queue) should be used when messages cannot be delivered by channels (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value might be:

MQUSEDLQ_NO
Messages that cannot be delivered by a channel will be treated as a failure and either the channel will discard them, or the channel will end, in accordance with the setting of NPMSPEED.

MQUSEDLQ_YES
If the queue manager DEADQ attribute provides the name of a dead-letter queue then it will be used, otherwise the behvaiour will be as for MQUSEDLQ_NO.

UserIdentifier **(MQCFST)**
Task user identifier (parameter identifier: MQCACH_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

XmitQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Inquire Channel Authentication Records:

The Inquire Channel Authentication Records (MQCMD_INQUIRE_CHLAUTH_RECS) command retrieves the allowed partner details and mappings to MCAUSER for a channel or set of channels.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

generic-channel-name **(MQCFST)**
The name of the channel or set of channels on which you are inquiring (parameter identifier: MQCACH_CHANNEL_NAME).

You can use the asterisk (*) as a wildcard to specify a set of channels, unless you set Match to MQMATCH_RUNCHECK. If you set Type to BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names.

Optional parameters

Address **(MQCFST)**
The IP address to be mapped (parameter identifier: MQCACH_CONNECTION_NAME).

This parameter is valid only when **Match** is MQMATCH_RUNCHECK and must not be generic.

ByteStringFilterCommand **(MQCFBF)**
Byte string filter command descriptor. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFBF - PCF byte string filter parameter" on page 1219 for information about using this filter condition.

If you specify a byte string filter, you cannot also specify an integer filter using the **IntegerFilterCommand** parameter, or a string filter using the **StringFilterCommand** parameter.

ChannelAuthAttrs **(MQCFIL)**
Authority record attributes (parameter identifier: MQIACF_CHLAUTH_ATTRS).

You can specify the following value in the attribute list on its own. This is the default value if the parameter is not specified.

MQIACF_ALL
All attributes.

If MQIACF_ALL is not specified, specify a combination of the following values:

MQCA_ALTERATION_DATE
Alteration Date.

MQCA_ALTERATION_TIME
Alteration Time.

MQCA_CHLAUTH_DESC
Description.

MQCA_CUSTOM
Custom.

MQCACH_CONNECTION_NAME
IP address filter.

MQCACH_MCA_USER_ID
MCA User ID mapped on the record.

MQIACH_USER_SOURCE
The source of the user ID for this record.

MQIACH_WARNING
Warning mode.

ClntUser **(MQCFST)**
The client asserted user ID to be matched (parameter identifier: MQCACH_CLIENT_USER_ID).

This parameter is valid only when **Match** is MQMATCH_RUNCHECK.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which the command was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFIF - PCF integer filter parameter” on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a byte string filter using the **ByteStringFilterCommand** parameter or a string filter using the **StringFilterCommand** parameter.

Match **(MQCFIN)**
Indicates the type of matching to be applied (parameter identifier MQIACH_MATCH). You can specify any one of the following values:

MQMATCH_RUNCHECK
A specific match is made against the supplied channel name and optionally supplied **Address**, **SSLPeer** , **QMName**, and **ClntUser** attributes to find the channel authentication record that will be matched by the channel at runtime if it connects into this queue manager. If the record discovered has **Warn** set to MQWARN_YES, a second record might also be displayed to show the

actual record the channel will use at runtime. The channel name supplied in this case cannot be generic. This option must be combined with **Type** MQCAUT_ALL.

MQMATCH_EXACT
Return only those records which exactly match the channel profile name supplied. If there are no asterisks in the channel profile name, this option returns the same output as MQMATCH_GENERIC.

MQMATCH_GENERIC
Any asterisks in the channel profile name are treated as wild cards. If there are no asterisks in the channel profile name, this returns the same output as MQMATCH_EXACT. For example, a profile of ABC* could result in records for ABC, ABC*, and ABCD being returned.

MQMATCH_ALL
Return all possible records that match the channel profile name supplied. If the channel name is generic in this case, all records that match the channel name are returned even if more specific matches exist. For example, a profile of SYSTEM.*.SVRCONN could result in records for SYSTEM.*, SYSTEM.DEF.*, SYSTEM.DEF.SVRCONN, and SYSTEM.ADMIN.SVRCONN being returned.

QMName **(MQCFST)**
The name of the remote partner queue manager to be matched (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

This parameter is valid only when **Match** is MQMATCH_RUNCHECK . The value cannot be generic.

SSLPeer **(MQCFST)**

The Distinguished Name of the certificate to be matched (parameter identifier: MQCACH_SSL_PEER_NAME).

This parameter is valid only when **Match** is MQMATCH_RUNCHECK .

The **SSLPeer** value is specified in the standard form used to specify a Distinguished Name and cannot be a generic value.

The maximum length of the parameter is MQ_SSL_PEER_NAME_LENGTH .

StringFilterCommand **(MQCFSF)**
String filter command descriptor. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify a byte string filter using the **ByteStringFilterCommand** parameter or an integer filter using the **IntegerFilterCommand** parameter.

Type **(MQCFIN)**
The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER (parameter identifier: MQIACF_CHLAUTH_TYPE). The following values are valid:

MQCAUT_BLOCKUSER
This channel authentication record prevents a specified user or users from connecting.

MQCAUT_BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses.

MQCAUT_SSLPEERMAP
This channel authentication record maps SSL Distinguished Names (DNs) to MCAUSER values.

MQCAUT_ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values.

MQCAUT_USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values.

MQCAUT_QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values.

MQCAUT_ALL
Inquire on all types of record. This is the default value.

Related information:

Channel authentication records

Inquire Channel Authentication Records (Response):

The response to the Inquire Channel Authentication Records (MQCMD_INQUIRE_CHLAUTH_RECS) command consists of the response header followed by the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned:
ChlAuth, *Type*, *Warn(yes)*

Always returned if type is MQCAUT_BLOCKUSER:
UserList

Always returned if type is MQCAUT_BLOCKADDR:
AddrList

Always returned if type is MQCAUT_SSLPEERMAP:
Address (unless blanks), *MCAUser (unless blanks)*, *SSLPeer*, *UserSrc*

Always returned if type is MQCAUT_ADDRESSMAP:
Address (unless blanks), *MCAUser (unless blanks)*, *UserSrc*

Always returned if type is MQCAUT_USERMAP:
Address (unless blanks), *CintUser*, *MCAUser (unless blanks)*, *UserSrc*

Always returned if type is MQCAUT_QMGRMAP:
Address (unless blanks), *MCAUser (unless blanks)*, *QMName*, *UserSrc*

Returned if requested:
Address, *AlterationDate*, *AlterationTime*, *Custom*, *Description*, *MCAUser*, *SSLPeer*, *UserSrc*, *Warn*

Response data

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss`.

Address **(MQCFST)**
The filter used to compare with the IP address of the partner queue manager or client at the other end of the channel (parameter identifier: MQCACH_CONNECTION_NAME).

AddrList **(MQCFSL)**
A list of up to 100 IP address patterns which are banned from accessing this queue manager on any channel (parameter identifier: MQCACH_CONNECTION_NAME_LIST).

Chlauth **(MQCFST)**
The name of the channel, or pattern that matches a set of channels, to which the channel authentication record applies (parameter identifier: MQCACH_CHANNEL_NAME).

Description **(MQCFST)**
Descriptive information about the channel authentication record (parameter identifier: MQCA_CHLAUTH_DESC).

ClntUser **(MQCFST)**
The client asserted user ID to be mapped to a new user ID, allowed through unchanged, or blocked (parameter identifier: MQCACH_CLIENT_USER_ID).

MCAUser **(MQCFST)**
The user identifier to be used when the inbound connection matches the SSL DN, IP address, client asserted user ID or remote queue manager name supplied (parameter identifier: MQCACH_MCA_USER_ID).

QMName **(MQCFST)**
The name of the remote partner queue manager to be mapped to a user ID, allowed through unchanged, or blocked (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

SSLPeer **(MQCFST)**
The filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel (parameter identifier: MQCACH_SSL_PEER_NAME).

Type **(MQCFIN)**
The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER (parameter identifier: MQIACF_CHLAUTH_TYPE). The following values can be returned:

MQCAUT_BLOCKUSER
This channel authentication record prevents a specified user or users from connecting.

MQCAUT_BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses.

MQCAUT_SSLPEERMAP
This channel authentication record maps SSL Distinguished Names (DNs) to MCAUSER values.

MQCAUT_ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values.

MQCAUT_USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values.

MQCAUT_QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values.

UserList **(MQCFSL)**
A list of up to 100 user IDs which are banned from use of this channel or set of channels (parameter identifier: MQCACH_MCA_USER_ID_LIST). Use the special value `*MQADMIN` to mean privileged or administrative users. The definition of this value depends on the operating system, as follows:

- On Windows, all members of the mqm group, the Administrators group and SYSTEM.
- On UNIX and Linux, all members of the mqm group.
- On IBM i, the profiles (users) qmqm and qmqmadm and all members of the qmqmadm group, and any user defined with the `*ALLOBJ` special setting.
- On z/OS, the user ID that the channel initiator and queue manager address spaces are running under.

UserSrc **(MQCFIN)**
The source of the user ID to be used for MCAUSER at run time (parameter identifier: MQIACH_USER_SOURCE).

The following values can be returned:

MQUSRC_MAP
Inbound connections that match this mapping use the user ID specified in the **MCAUser** attribute.

MQUSRC_NOACCESS
Inbound connections that match this mapping have no access to the queue manager and the channel ends immediately.

MQUSRC_CHANNEL
Inbound connections that match this mapping use the flowed user ID or any user defined on the channel object in the MCAUSER field.

Warn **(MQCFIN)**
Indicates whether this record operates in warning mode (parameter identifier: MQIACH_WARNING).

MQWARN_NO
This record does not operate in warning mode. Any inbound connection that matches this record is blocked. This is the default value.

MQWARN_YES
This record operates in warning mode. Any inbound connection that matches this record and would therefore be blocked is allowed access. An error message is written and, if events are configured, an event message is created showing the details of what would have been blocked. The connection is allowed to continue.

Inquire Channel Listener:

The Inquire Channel Listener (MQCMD_INQUIRE_LISTENER) command inquires about the attributes of existing WebSphere MQ listeners.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ListenerName **(MQCFST)**
Listener name (parameter identifier: MQCACH_LISTENER_NAME).

This parameter is the name of the listener with attributes that are required. Generic listener names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all listeners having names that start with the selected character string. An asterisk on its own matches all possible names.

The listener name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ListenerAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFIF - PCF integer filter parameter” on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

ListenerAttrs **(MQCFIL)**
Listener attributes (parameter identifier: MQIACF_LISTENER_ATTRS).

The attribute list might specify the following value on its own- default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
Date on which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCACH_IP_ADDRESS
IP address for the listener.

MQCACH_LISTENER_DESC
Description of listener definition.

MQCACH_LISTENER_NAME
Name of listener definition.

MQCACH_LOCAL_NAME
NetBIOS local name that the listener uses. MQCACH_LOCAL_NAME is valid only on Windows.

MQCACH_TP_NAME
The LU 6.2 transaction program name. MQCACH_TP_NAME is valid only on Windows.

MQIACH_ADAPTER
Adapter number on which NetBIOS listens. MQIACH_ADAPTER is valid only on Windows.

MQIACH_BACKLOG
Number of concurrent connection requests that the listener supports.

MQIACH_COMMAND_COUNT
Number of commands that the listener can use. MQIACH_COMMAND_COUNT is valid only on Windows.

MQIACH_LISTENER_CONTROL
Specifies when the queue manager starts and stops the listener.

MQIACH_NAME_COUNT
Number of names that the listener can use. MQIACH_NAME_COUNT is valid only on Windows.

MQIACH_PORT
Port number.

MQIACH_SESSION_COUNT
Number of sessions that the listener can use. MQIACH_SESSION_COUNT is valid only on Windows.

MQIACH_SOCKET
SPX socket on which to listen. MQIACH_SOCKET is valid only on Windows.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ListenerAttrs* except MQCACH_LISTENER_NAME. Use this parameter to restrict the output from

the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

TransportType **(MQCFIN)**
Transport protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

If you specify this parameter, information is returned relating only to those listeners defined with the specified transport protocol type. If you specify an attribute in the *ListenerAttrs* list which is valid only for listeners of a different transport protocol type, it is ignored and no error is raised. If you specify this parameter, it must occur immediately after the *ListenerName* parameter.

If you do not specify this parameter, or if you specify it with a value of MQXPT_ALL, information about all listeners is returned. Valid attributes in the *ListenerAttrs* list which are not applicable to the listener are ignored, and no error messages are issued. The value can be:

MQXPT_ALL
All transport types.

MQXPT_LU62
SNA LU 6.2. MQXPT_LU62 is valid only on Windows.

MQXPT_NETBIOS
NetBIOS. MQXPT_NETBIOS is valid only on Windows.

MQXPT_SPX
SPX. MQXPT_SPX is valid only on Windows.

MQXPT_TCP
Transmission Control Protocol/Internet Protocol (TCP/IP).

Inquire Channel Listener (Response):

The response to the Inquire Channel Listener (MQCMD_INQUIRE_LISTENER) command consists of the response header followed by the *ListenerName* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

If a generic listener name was specified, one such message is generated for each listener found.

Always returned:
ListenerName

Returned if requested:
Adapter, *AlterationDate*, *AlterationTime*, *Backlog*, *Commands*, *IPAddress*, *ListenerDesc*, *LocalName*, *NetbiosNames*, *Port*, *Sessions*, *Socket*, *StartMode*, *TPname*, *TransportType*

Response data

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date, in the form `yyyy-mm-dd`, on which the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time, in the form `hh.mm.ss`, at which the information was last altered.

Adapter **(MQCFIN)**
Adapter number (parameter identifier: MQIACH_ADAPTER).

The adapter number on which NetBIOS listens. This parameter is valid only on Windows.

Backlog **(MQCFIN)**
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

Commands **(MQCFIN)**
Adapter number (parameter identifier: MQIACH_COMMAND_COUNT).

The number of commands that the listener can use. This parameter is valid only on Windows.

IPAddress **(MQCFST)**
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form.

The maximum length of the string is MQ_CONN_NAME_LENGTH

ListenerDesc **(MQCFST)**
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

ListenerName **(MQCFST)**
Name of listener definition (parameter identifier: MQCACH_LISTENER_NAME).

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

LocalName **(MQCFST)**
NetBIOS local name (parameter identifier: MQCACH_LOCAL_NAME).

The NetBIOS local name that the listener uses. This parameter is valid only on Windows.

The maximum length of the string is MQ_CONN_NAME_LENGTH

NetbiosNames **(MQCFIN)**
NetBIOS names (parameter identifier: MQIACH_NAME_COUNT).

The number of names that the listener supports. This parameter is valid only on Windows.

Port **(MQCFIN)**
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP. This parameter is valid only if the value of *TransportType* is MQXPT_TCP.

Sessions **(MQCFIN)**
NetBIOS sessions (parameter identifier: MQIACH_SESSION_COUNT).

The number of sessions that the listener can use. This parameter is valid only on Windows.

Socket **(MQCFIN)**
SPX socket number (parameter identifier: MQIACH_SOCKET).

The SPX socket on which to listen. This parameter is valid only if the value of *TransportType* is MQXPT_SPX.

StartMode **(MQCFIN)**
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. MQSVC_CONTROL_MANUAL is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The listener is to be started at the same time as the queue manager is started, but is not request to stop when the queue manager is stopped.

TPName **(MQCFST)**
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The LU 6.2 transaction program name. This parameter is valid only on Windows.

The maximum length of the string is MQ_TP_NAME_LENGTH

TransportType **(MQCFIN)**
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_TCP
TCP.

MQXPT_LU62
LU 6.2. MQXPT_LU62 is valid only on Windows.

MQXPT_NETBIOS
NetBIOS. MQXPT_NETBIOS is valid only on Windows.

MQXPT_SPX
SPX. MQXPT_SPX is valid only on Windows.

Inquire Channel Listener Status:

The Inquire Channel Listener Status (MQCMD_INQUIRE_LISTENER_STATUS) command inquires about the status of one or more WebSphere MQ listener instances.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

You must specify the name of a listener for which you want to receive status information. You can specify a listener by using either a specific listener name or a generic listener name. By using a generic listener name, you can display either:

- Status information for all listener definitions, by using a single asterisk (*), or
- Status information for one or more listeners that match the specified name.

Required parameters

ListenerName **(MQCFST)**
Listener name (parameter identifier: MQCACH_LISTENER_NAME).

Generic listener names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all listeners having names that start with the selected character string. An asterisk on its own matches all possible names.

The listener name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ListenerStatusAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

ListenerStatusAttrs **(MQCFIL)**
Listener status attributes (parameter identifier: MQIACF_LISTENER_STATUS_ATTRS).

The attribute list can specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCACH_IP_ADDRESS
IP address of the listener.

MQCACH_LISTENER_DESC
Description of listener definition.

MQCACH_LISTENER_NAME
Name of listener definition.

MQCACH_LISTENER_START_DATE
The date on which the listener was started.

MQCACH_LISTENER_START_TIME
The time at which the listener was started.

MQCACH_LOCAL_NAME
NetBIOS local name that the listener uses. MQCACH_LOCAL_NAME is valid only on Windows.

MQCACH_TP_NAME
LU6.2 transaction program name. MQCACH_TP_NAME is valid only on Windows.

MQIACF_PROCESS_ID
Operating system process identifier associated with the listener.

MQIACH_ADAPTER
Adapter number on which NetBIOS listens. MQIACH_ADAPTER is valid only on Windows.

MQIACH_BACKLOG
Number of concurrent connection requests that the listener supports.

MQIACH_COMMAND_COUNT
Number of commands that the listener can use. MQIACH_COMMAND_COUNT is valid only on Windows.

MQIACH_LISTENER_CONTROL
How the listener is to be started and stopped.

MQIACH_LISTENER_STATUS
Status of the listener.

MQIACH_NAME_COUNT
Number of names that the listener can use. MQIACH_NAME_COUNT is valid only on Windows.

MQIACH_PORT
Port number for TCP/IP.

MQIACH_SESSION_COUNT
Number of sessions that the listener can use. MQIACH_SESSION_COUNT is valid only on Windows.

MQIACH_SOCKET
SPX socket. MQIACH_SOCKET is valid only on Windows.

MQIACH_XMIT_PROTOCOL_TYPE
Transport type.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ListenerStatusAttrs* except MQCACH_LISTENER_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Error code

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_LSTR_STATUS_NOT_FOUND
Listener status not found.

Inquire Channel Listener Status (Response):

The response to the Inquire Channel Listener Status (MQCMD_INQUIRE_LISTENER_STATUS) command consists of the response header followed by the *ListenerName* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

If a generic listener name was specified, one such message is generated for each listener found.

Always returned:
ListenerName

Returned if requested:
Adapter, *Backlog*, *ChannelCount*, *Commands*, *IPAddress*, *ListenerDesc*, *LocalName*, *NetbiosNames*, *Port*, *ProcessId*, *Sessions*, *Socket*, *StartDate*, *StartMode*, *StartTime*, *Status*, *TPname*, *TransportType*

Response data

Adapter **(MQCFIN)**
Adapter number (parameter identifier: MQIACH_ADAPTER).

The adapter number on which NetBIOS listens.

Backlog **(MQCFIN)**
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

Commands **(MQCFIN)**
Adapter number (parameter identifier: MQIACH_COMMAND_COUNT).

The number of commands that the listener can use.

IPAddress **(MQCFST)**
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form.

The maximum length of the string is MQ_CONN_NAME_LENGTH

ListenerDesc **(MQCFST)**
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

The maximum length of the string is MQ_LISTENER_DESC_LENGTH.

ListenerName **(MQCFST)**
Name of listener definition (parameter identifier: MQCACH_LISTENER_NAME).

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

LocalName **(MQCFST)**
NetBIOS local name (parameter identifier: MQCACH_LOCAL_NAME).

The NetBIOS local name that the listener uses.

The maximum length of the string is MQ_CONN_NAME_LENGTH

NetbiosNames **(MQCFIN)**
NetBIOS names (parameter identifier: MQIACH_NAME_COUNT).

The number of names that the listener supports.

Port **(MQCFIN)**
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP.

ProcessId **(MQCFIN)**
Process identifier (parameter identifier: MQIACF_PROCESS_ID).

The operating system process identifier associated with the listener.

Sessions **(MQCFIN)**
NetBIOS sessions (parameter identifier: MQIACH_SESSION_COUNT).

The number of sessions that the listener can use.

Socket **(MQCFIN)**
SPX socket number (parameter identifier: MQIACH_SOCKET).

The SPX socket on which the listener is to listen.

StartDate **(MQCFST)**
Start date (parameter identifier: MQCACH_LISTENER_START_DATE).

The date, in the form `yyyy-mm-dd`, on which the listener was started.

The maximum length of the string is MQ_DATE_LENGTH

StartMode **(MQCFIN)**
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The listener is not to be started automatically or stopped automatically. It is to be controlled by user command. MQSVC_CONTROL_MANUAL is the default value.

MQSVC_CONTROL_Q_MGR
The listener being defined is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The listener is to be started at the same time as the queue manager is started, but is not request to stop when the queue manager is stopped.

StartTime **(MQCFST)**
Start date (parameter identifier: MQCACH_LISTENER_START_TIME).

The time, in the form `hh.mm.ss`, at which the listener was started.

The maximum length of the string is MQ_TIME_LENGTH

Status **(MQCFIN)**
Listener status (parameter identifier: MQIACH_LISTENER_STATUS).

The status of the listener. The value can be:

MQSVC_STATUS_STARTING
The listener is in the process of initializing.

MQSVC_STATUS_RUNNING
The listener is running.

MQSVC_STATUS_STOPPING
The listener is stopping.

TPName **(MQCFST)**
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The LU 6.2 transaction program name.

The maximum length of the string is MQ_TP_NAME_LENGTH

TransportType **(MQCFIN)**
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_TCP
TCP.

MQXPT_LU62
LU 6.2. MQXPT_LU62 is valid only on Windows.

MQXPT_NETBIOS
NetBIOS. MQXPT_NETBIOS is valid only on Windows.

MQXPT_SPX
SPX. MQXPT_SPX is valid only on Windows.

Inquire Channel Names:

The Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command inquires a list of WebSphere MQ channel names that match the generic channel name, and the optional channel type specified.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If present, this parameter limits the channel names returned to channels of the specified type.

The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

MQCHT_ALL
All types.

The default value if this parameter is not specified is MQCHT_ALL, which means that channels of all types except MQCHT_CLNTCONN are eligible.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Error code

This command might return the following error code in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

Inquire Channel Names (Response):

The response to the Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command consists of one response per client connection channel (except for SYSTEM.DEF.CLNTCONN), and a final message with all the remaining channels.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Always returned:
ChannelNames, *ChannelTypes*

Returned if requested:
None

On z/OS only, one additional parameter structure (with the same number of entries as the *ChannelNames* structure), is returned. Each entry in the structure, *QSGDispositions*, indicates the disposition of the object with the corresponding entry in the *ChannelNames* structure.

Response data

ChannelNames **(MQCFSL)**
List of channel names (parameter identifier: MQCACH_CHANNEL_NAMES).

ChannelTypes **(MQCFIL)**
List of channel types (parameter identifier: MQIACH_CHANNEL_TYPES). Possible values for fields in this structure are those values permitted for the *ChannelType* parameter, except MQCHT_ALL.

QSGDispositions **(MQCFIL)**
List of QSG dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter is valid only on z/OS. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Channel Status:

The Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command inquires about the status of one or more channel instances.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

You must specify the name of the channel for which you want to inquire status information. This name can be a specific channel name or a generic channel name. By using a generic channel name, you can inquire either:

- Status information for all channels, or
- Status information for one or more channels that match the specified name.

You must also specify whether you want:

- The status data (of current channels only), or
- The saved status data of all channels, or
- On z/OS only, the short status data of the channel.

  Status for all channels that meet the selection criteria is returned, whether the channels were defined manually or automatically.

This command includes a check on the current depth of the transmission queue for the channel, if the channel is a CLUSSDR channel. To issue this command, you must be authorized to inquire the queue depth, and to do this requires *+inq* authority on the transmission queue. Note that another name for this authority is MQZAO_INQUIRE.

Without this authority this command fails with a reason code of MQRC_NOT_AUTHORIZED.

There are three classes of data available for channel status. These classes are **saved**, **current**, and **short**. The status fields available for saved data are a subset of the fields available for current data and are called **common** status fields. Although the common data *fields* are the same, the data *values* might be different for saved and current status. The rest of the fields available for current data are called **current-only** status fields.

- **Saved** data consists of the common status fields. This data is reset at the following times:
  - For all channels:
    - When the channel enters or leaves STOPPED or RETRY state
  - For a sending channel:
    - Before requesting confirmation that a batch of messages has been received
    - When confirmation has been received
  - For a receiving channel:
    - Just before confirming that a batch of messages has been received
  - For a server connection channel:
    - No data is saved

  Therefore, a channel which has never been current does not have any saved status.
- **Current** data consists of the common status fields and current-only status fields. The data fields are continually updated as messages are sent or received.
- **Short** data consists of the queue manager name that owns the channel instance. This class of data is available only on z/OS.

This method of operation has the following consequences:

- An inactive channel might not have any saved status if it has never been current or has not yet reached a point where saved status is reset.
- The “common” data fields might have different values for saved and current status.
- A current channel always has current status and might have saved status.

Channels can be current or inactive:

Current channels
: These are channels that have been started, or on which a client has connected, and that have not finished or disconnected normally. They might not yet have reached the point of transferring messages, or data, or even of establishing contact with the partner. Current channels have **current** status and can also have **saved** or **short** status.

  The term **Active** is used to describe the set of current channels which are not stopped.

Inactive channels
: These are channels that have either not been started or on which a client has not connected, or

that have finished or disconnected normally. (If a channel is stopped, it is not yet considered to have finished normally and is, therefore, still current.) Inactive channels have either **saved** status or no status at all.

There can be more than one instance of a receiver, requester, cluster-sender, cluster-receiver, or server-connection channel current at the same time (the requester is acting as a receiver). This situation occurs if several senders, at different queue managers, each initiate a session with this receiver, using the same channel name. For channels of other types, there can only be one instance current at any time.

For all channel types, however, there can be more than one set of saved status information available for a particular channel name. At most one of these sets relates to a current instance of the channel, the rest relate to previously current instances. Multiple instances arise if different transmission queue names or connection names have been used with the same channel. This situation can happen in the following cases:

- At a sender or server:
  - If the same channel has been connected to by different requesters (servers only),
  - If the transmission queue name has been changed in the definition, or
  - If the connection name has been changed in the definition.
- At a receiver or requester:
  - If the same channel has been connected to by different senders or servers, or
  - If the connection name has been changed in the definition (for requester channels initiating connection).

The number of sets returned for a particular channel can be limited by using the *XmitQName*, *ConnectionName* and *ChannelInstanceType* parameters.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The channel name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

MaxResponses **(MQCFIN)**
The maximum number of clients to return status for. This parameter is optional for all channels.

ResponseRestartPoint **(MQCFIN)**
The first client to return status for. The combination of this parameter with **MaxResponses** enables the range of clients to be specified. This parameter is optional for all other channels.

Optional parameters

ChannelDisposition **(MQCFIN)**
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels for which information is to be returned. The value can be:

MQCHLD_ALL
Returns requested status information for private channels.

In a shared queue environment where the command is being executed on the queue manager where it was issued, or if *ChannelInstanceType* has a value of MQOT_CURRENT_CHANNEL, this option also displays the requested status information for shared channels.

MQCHLD_PRIVATE
Returns requested status information for private channels.

MQCHLD_SHARED
Returns requested status information for shared channels.

The status information that is returned for various combinations of *ChannelDisposition*, *CommandScope*, and status type, is summarized in Table 96, Table 97, and Table 98 on page 995.

Table 96. ChannelDisposition and CommandScope for Inquire Channel Status, Current

ChannelDisposition	*CommandScope* **blank or local queue manager**	*CommandScope(qmgr-name)*	*CommandScope(*)*
MQCHLD_PRIVATE	Common and current-only status for current private channels on the local queue manager	Common and current-only status for current private channels on the named queue manager	Common and current-only status for current private channels on all queue managers
MQCHLD_SHARED	Common and current-only status for current shared channels on the local queue manager	Common and current-only status for current shared channels on the named queue manager	Common and current-only status for current shared channels on all queue managers
MQCHLD_ALL	Common and current-only status for current private and shared channels on the local queue manager	Common and current-only status for current private and shared channels on the named queue manager	Common and current-only status for current private and shared channels on all active queue managers

Table 97. ChannelDisposition and CommandScope for Inquire Channel Status, Short

ChannelDisposition	*CommandScope* **blank or local queue manager**	*CommandScope(qmgr-name)*	*CommandScope(*)*
MQCHLD_PRIVATE	*ChannelStatus* and short status for current private channels on the local queue manager	*ChannelStatus* and short status for current private channels on the named queue manager	*ChannelStatus* and short status for current private channels on all active queue managers
MQCHLD_SHARED	*ChannelStatus* and short status for current shared channels on all active queue managers in the queue-sharing group	Not permitted	Not permitted
MQCHLD_ALL	*ChannelStatus* and short status for current private channels on the local queue manager and current shared channels in the queue-sharing group(1)	*ChannelStatus* and short status for current private channels on the named queue manager	*ChannelStatus* and short status for current private, and shared, channels on all active queue managers in the queue-sharing group(1)

Note:

1. In this case you get two separate sets of responses to the command on the queue manager where it was entered; one for MQCHLD_PRIVATE and one for MQCHLD_SHARED.

Table 98. ChannelDisposition and CommandScope for Inquire Channel Status, Saved

ChannelDisposition	*CommandScope* **blank or local queue manager**	*CommandScope(qmgr-name)*	*CommandScope(*)*
MQCHLD_PRIVATE	Common status for saved private channels on the local queue manager	Common status for saved private channels on the named queue manager	Common status for saved private channels on all active queue managers
MQCHLD_SHARED	Common status for saved shared channels on all active queue managers in the queue-sharing group	Not permitted	Not permitted
MQCHLD_ALL	Common status for saved private channels on the local queue manager and saved shared channels in the queue-sharing group	Common status for saved private channels on the named queue manager	Common status for saved private, and shared, channels on all active queue managers in the queue-sharing group

You cannot use this parameter as a filter keyword.

ClientIdentifer **(MQCFST)**
The ClientId of the client.

MaxResponses **(MQCFIN)**
The maximum number of clients to return status for.

ResponseRestartPoint **(MQCFIN)**
The first client to return status for. The combination of this parameter with **MaxResponses** enables the range of clients to be specified.

ChannelInstanceAttrs **(MQCFIL)**
Channel instance attributes (parameter identifier: MQIACH_CHANNEL_INSTANCE_ATTRS).

If status information is requested which is not relevant for the particular channel type, it is not an error. Similarly, it is not an error to request status information that is applicable only to active channels for saved channel instances. In both of these cases, no structure is returned in the response for the information concerned.

For a saved channel instance, the MQCACH_CURRENT_LUWID, MQIACH_CURRENT_MSGS, and MQIACH_CURRENT_SEQ_NUMBER attributes have meaningful information only if the channel instance is in doubt. However, the attribute values are still returned when requested, even if the channel instance is not in-doubt.

The attribute list might specify the following value on its own:

MQIACF_ALL
All attributes.

MQIACF_ALL is the default value used if the parameter is not specified or it can specify a combination of the following:

- Relevant for common status:

  The following information applies to all sets of channel status, whether the set is current.

  MQCACH_CHANNEL_NAME
  Channel name.

  MQCACH_CONNECTION_NAME
  Connection name.

  MQCACH_CURRENT_LUWID
  Logical unit of work identifier for current batch.

MQCACH_LAST_LUWID
Logical unit of work identifier for last committed batch.

MQCACH_XMIT_Q_NAME
Transmission queue name.

MQIACH_CHANNEL_INSTANCE_TYPE
Channel instance type.

MQIACH_CHANNEL_TYPE
Channel type.

MQIACH_CURRENT_MSGS
Number of messages sent or received in current batch.

MQIACH_CURRENT_SEQ_NUMBER
Sequence number of last message sent or received.

MQIACH_INDOUBT_STATUS
Whether the channel is currently in-doubt.

MQIACH_LAST_SEQ_NUMBER
Sequence number of last message in last committed batch.

MQCACH_CURRENT_LUWID, MQCACH_LAST_LUWID, MQIACH_CURRENT_MSGS, MQIACH_CURRENT_SEQ_NUMBER, MQIACH_INDOUBT_STATUS and MQIACH_LAST_SEQ_NUMBER do not apply to server-connection channels, and no values are returned. If specified on the command, they are ignored.

- Relevant for current-only status:

  The following information applies only to current channel instances. The information applies to all channel types, except where stated.

  MQCA_Q_MGR_NAME
  Name of the queue manager that owns the channel instance. This parameter is valid only on z/OS.

  MQCA_REMOTE_Q_MGR_NAME
  Queue manager name, or queue-sharing group name of the remote system. The remote queue manager name is always returned regardless of the instance attributes requested.

  MQCACH_CHANNEL_START_DATE
  Date channel was started.

  MQCACH_CHANNEL_START_TIME
  Time channel was started.

  MQCACH_LAST_MSG_DATE
  Date last message was sent, or MQI call was handled.

  MQCACH_LAST_MSG_TIME
  Time last message was sent, or MQI call was handled.

  MQCACH_LOCAL_ADDRESS
  Local communications address for the channel.

  MQCACH_MCA_JOB_NAME
  Name of MCA job.

  This parameter is not valid on z/OS.

  You cannot use MQCACH_MCA_JOB_NAME as a parameter to filter on.

  MQCACH_MCA_USER_ID
  The user ID used by the MCA.

MQCACH_REMOTE_APPL_TAG
Remote partner application name. MQCACH_REMOTE_APPL_TAG is the name of the client application at the remote end of the channel. This parameter applies only to server-connection channels.

MQCACH_REMOTE_PRODUCT
Remote partner product identifier. This is the product identifier of the IBM WebSphere MQ code running at the remote end of the channel.

MQCACH_REMOTE_VERSION
Remote partner version. This is the version of the IBM WebSphere MQ code running at the remote end of the channel.

MQCACH_SSL_SHORT_PEER_NAME
SSL short peer name.

MQCACH_SSL_CERT_ISSUER_NAME
The full Distinguished Name of the issuer of the remote certificate.

MQCACH_SSL_CERT_USER_ID
User ID associated with the remote certificate. MQCACH_SSL_CERT_USER_ID is valid on z/OS only.

MQIA_MONITORING_CHANNEL
The level of monitoring data collection.

MQIACF_MONITORING
All channel status monitoring attributes. These attributes are:

MQIA_MONITORING_CHANNEL
The level of monitoring data collection.

MQIACH_BATCH_SIZE_INDICATOR
Batch size.

MQIACH_COMPRESSION_RATE
The compression rate achieved displayed to the nearest percentage.

MQIACH_COMPRESSION_TIME
The amount of time per message, displayed in microseconds, spent during compression or decompression.

MQIACH_EXIT_TIME_INDICATOR
Exit time.

MQIACH_NETWORK_TIME_INDICATOR
Network time.

MQIACH_XMITQ_MSGS_AVAILABLE
Number of messages available to the channel on the transmission queue.

MQIACH_XMITQ_TIME_INDICATOR
Time on transmission queue.

You cannot use MQIACF_MONITORING as a parameter to filter on.

MQIACH_BATCH_SIZE_INDICATOR
Batch size.

You cannot use MQIACH_BATCH_SIZE_INDICATOR as a parameter to filter on.

MQIACH_BATCHES
Number of completed batches.

MQIACH_BUFFERS_RCVD
Number of buffers received.

MQIACH_BUFFERS_SENT
Number of buffers sent.

MQIACH_BYTES_RCVD
Number of bytes received.

MQIACH_BYTES_SENT
Number of bytes sent.

MQIACH_CHANNEL_SUBSTATE
The channel substate.

MQIACH_COMPRESSION_RATE
The compression rate achieved displayed to the nearest percentage.

You cannot use MQIACH_COMPRESSION_RATE as a parameter to filter on.

MQIACH_COMPRESSION_TIME
The amount of time per message, displayed in microseconds, spent during compression or decompression.

You cannot use MQIACH_COMPRESSION_TIME as a parameter to filter on.

MQIACH_CURRENT_SHARING_CONVS
Requests information about the current number of conversations on this channel instance.

This attribute applies only to TCP/IP server-connection channels.

MQIACH_EXIT_TIME_INDICATOR
Exit time.

You cannot use MQIACH_EXIT_TIME_INDICATOR as a parameter to filter on.

MQIACH_HDR_COMPRESSION
Technique used to compress the header data sent by the channel.

MQIACH_KEEP_ALIVE_INTERVAL
The KeepAlive interval in use for this session. This parameter is significant only for z/OS.

MQIACH_LONG_RETRIES_LEFT
Number of long retry attempts remaining.

MQIACH_MAX_MSG_LENGTH
Maximum message length. MQIACH_MAX_MSG_LENGTH is valid only on z/OS.

MQIACH_MAX_SHARING_CONVS
Requests information about the maximum number of conversations on this channel instance.

This attribute applies only to TCP/IP server-connection channels.

MQIACH_MCA_STATUS
MCA status.

You cannot use MQIACH_MCA_STATUS as a parameter to filter on.

MQIACH_MSG_COMPRESSION
Technique used to compress the message data sent by the channel.

MQIACH_MSGS
Number of messages sent or received, or number of MQI calls handled.

MQIACH_NETWORK_TIME_INDICATOR
Network time.

You cannot use MQIACH_NETWORK_TIME_INDICATOR as a parameter on which to filter.

MQIACH_SHORT_RETRIES_LEFT
Number of short retry attempts remaining.

MQIACH_SSL_KEY_RESETS
Number of successful SSL key resets.

MQIACH_SSL_RESET_DATE
Date of previous successful SSL secret key reset.

MQIACH_SSL_RESET_TIME
Time of previous successful SSL secret key reset.

MQIACH_STOP_REQUESTED
Whether user stop request has been received.

MQIACH_XMITQ_MSGS_AVAILABLE
Number of messages available to the channel on the transmission queue.

MQIACH_XMITQ_TIME_INDICATOR
Time on transmission queue.

You cannot use MQIACH_XMITQ_TIME_INDICATOR as a parameter to filter on.

The following value is supported on all platforms:

MQIACH_BATCH_SIZE
Batch size.

The following value is supported on all platforms:

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

The following attributes do not apply to server-connection channels, and no values are returned. If specified on the command they are ignored:

- MQIACH_BATCH_SIZE_INDICATOR
- MQIACH_BATCH_SIZE
- MQIACH_BATCHES
- MQIACH_LONG_RETRIES_LEFT
- MQIACH_NETWORK_TIME
- MQIACH_NPM_SPEED
- MQCA_REMOTE_Q_MGR_NAME
- MQIACH_SHORT_RETRIES_LEFT
- MQIACH_XMITQ_MSGS_AVAILABLE
- MQIACH_XMITQ_TIME_INDICATOR

The following attributes apply only to server-connection channels. If specified on the command for other types of channel the attribute is ignored and no value is returned:

- MQIACH_CURRENT_SHARING_CONVS
- MQIACH_MAX_SHARING_CONVS

- Relevant for short status:

  The following parameter applies to current channels on z/OS:

  MQCACH_Q_MGR_NAME
  Name of the queue manager that owns the channel instance.

ChannelInstanceType **(MQCFIN)**
Channel instance type (parameter identifier: MQIACH_CHANNEL_INSTANCE_TYPE).

It is always returned regardless of the channel instance attributes requested.

The value can be:

MQOT_CURRENT_CHANNEL
The channel status.

MQOT_CURRENT_CHANNEL is the default, and indicates that only current status information for active channels is to be returned.

Both common status information and active-only status information can be requested for current channels.

MQOT_SAVED_CHANNEL
Saved channel status.

Specify MQOT_SAVED_CHANNEL to cause saved status information for both active and inactive channels to be returned.

Only common status information can be returned. Active-only status information is not returned for active channels if this keyword is specified.

MQOT_SHORT_CHANNEL
Short channel status (valid on z/OS only).

Specify MQOT_SHORT_CHANNEL to cause short status information for current channels to be returned.

Other common status and current-only status information are not returned for current channels if this keyword is specified.

You cannot use MQIACH_CHANNEL_INSTANCE_TYPE as a parameter to filter on.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

ConnectionName **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

If this parameter is present, eligible channel instances are limited to those using this connection name. If it is not specified, eligible channel instances are not limited in this way.

The connection name is always returned, regardless of the instance attributes requested.

The value returned for *ConnectionName* might not be the same as in the channel definition, and might differ between the current channel status and the saved channel status. (Using *ConnectionName* for limiting the number of sets of status is therefore not recommended.)

For example, when using TCP, if *ConnectionName* in the channel definition:

- Is blank or is in *host name* format, the channel status value has the resolved IP address.
- Includes the port number, the current channel status value includes the port number (except on z/OS), but the saved channel status value does not.

The maximum length of the string is MQ_CONN_NAME_LENGTH.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ChannelInstanceAttrs* except MQIACF_ALL and others as noted. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ChannelInstanceAttrs* except MQCACH_CHANNEL_NAME and others as noted. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter for *ConnectionName* or *XmitQName*, you cannot also specify the *ConnectionName* or *XmitQName* parameter.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

XmitQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

If this parameter is present, eligible channel instances are limited to those using this transmission queue. If it is not specified, eligible channel instances are not limited in this way.

The transmission queue name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Error code

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHL_INST_TYPE_ERROR
Channel instance type not valid.

MQRCCF_CHL_STATUS_NOT_FOUND
Channel status not found.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

Inquire Channel Status (MQTT):

The Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) (MQTT) command inquires about the status of one or more Telemetry channel instances.

You must specify the name of the channel for which you want to inquire status information. This name can be a specific channel name or a generic channel name. By using a generic channel name, you can inquire either:

- Status information for all channels, or
- Status information for one or more channels that match the specified name.

Note: The **Inquire Channel Status** command for IBM WebSphere MQ Telemetry has the potential to return a far larger number of responses than if the command was run for a IBM WebSphere MQ channel. For this reason, the IBM WebSphere MQ Telemetry server does not return more responses than fit on the reply-to queue. The number of responses is limited to the value of MAXDEPTH parameter of the `SYSTEM.MQSC.REPLY.QUEUE` queue. When a IBM WebSphere MQ Telemetry command is truncated by the IBM WebSphere MQ Telemetry server, the AMQ8492 message is displayed specifying how many responses are returned based on the size of MAXDEPTH.

If the **ClientIdentifier** parameter is not specified, the output of the **Inquire Channel Status** command is a summary of statuses of all clients connected to the channel. One PCF response message is returned per channel.

If the **ClientIdentifier** parameter is specified, separate PCF response messages are returned for each client connection. The **ClientIdentifier** parameter may be a wildcard, in which the status for all clients that match the **ClientIdentifier** string is returned (within the limits of **MaxResponses** and **ResponseRestartPoint** if they are set).

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example `ABC*`, and it selects all objects which have names that start with the selected character string. An asterisk on its own matches all possible names.

This parameter is allowed for only when the **ResponseType** parameter is set to MQRESP_TOTAL.

The channel name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value must be:

MQCHT_MQTT
Telemetry.

Optional parameters

ClientIdentifer **(MQCFST)**
The ClientId of the client (parameter identifier: MQCACH_CLIENT_ID).

MaxResponses **(MQCFIN)**
The maximum number of clients to return status for (parameter identifier: MQIA_MAX_RESPONSES).

This parameter is only allowed when the **ClientIdentifier** parameter is specified.

ResponseRestartPoint **(MQCFIN)**
The first client to return status for (parameter identifier: MQIA_RESPONSE_RESTART_POINT). The combination of this parameter with **MaxResponses** enables the range of clients to be specified.

This parameter is only allowed when the **ClientIdentifier** parameter is specified.

Client details mode

STATUS
The current status of the client (parameter identifier: MQIACH_CHANNEL_STATUS).

CONNAME
The name of the remote connection (ip address) (parameter identifier: MQCACH_CONNECTION_NAME).

KAINT
The keep alive interval of the client (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

MCANAME
Message channel agent name (parameter identifier: MQCACH_MCA_USER_ID).

MSGSNT
Number of messages sent by the client since it last connected (parameter identifier: MQIACH_MSGS_SENT).

MSGRCVD
Number of messages received by the client since it last connected (parameter identifier: MQIACH_MSGS_RECEIVED / MQIACH_MSGS_RCVD).

INDOUBTIN
Number of in doubt, inbound messages to the client (parameter identifier: MQIACH_IN_DOUBT_IN).

INDOUBTOUT
Number of in doubt, outbound messages to the client (parameter identifier: MQIACH_IN_DOUBT_OUT).

PENDING
Number of outbound pending messages (parameter identifier: MQIACH_PENDING_OUT).

LMSGDATE
Date last message was received or sent (parameter identifier: MQCACH_LAST_MSG_DATE).

LMSGTIME
Time last message was received or sent (parameter identifier: MQCACH_LAST_MSG_TIME).

CHLSDATE
Date channel started (parameter identifier: MQCACH_CHANNEL_START_DATE).

CHLSTIME
Time channel was started (parameter identifier: MQCACH_CHANNEL_START_TIME).

Error code

This command might return the following error code in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHL_INST_TYPE_ERROR
Channel instance type not valid.

MQRCCF_CHL_STATUS_NOT_FOUND
Channel status not found.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

Inquire Channel Status (Response):

The response to the Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command consists of the response header followed by several structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

These structures are

- The *ChannelName* structure,
- The *ChannelDisposition* structure (on z/OS only),
- The *ChannelInstanceType* structure
- The *ChannelStatus* structure (except on z/OS channels whose *ChannelInstanceType* parameter has a value of MQOT_SAVED_CHANNEL.
- The *ChannelType* structure
- The *ConnectionName* structure
- The *RemoteApplTag* structure
- The *RemoteQMgrName* structure
- The *StopRequested* structure
- The *XmitQName* structure

which are then followed by the requested combination of status attribute parameter structures. One such message is generated for each channel instance found that matches the criteria specified on the command.

On z/OS, if the value for any of these parameters exceeds 999999999, it is returned as 999999999:

- *Batches*
- *BuffersReceived*
- *BuffersSent*
- *BytesReceived*
- *BytesSent*
- *CompressionTime*
- *CurrentMsgs*
- *ExitTime*
- *Msgs*
- *NetTime*
- *SSLKeyResets*
- *XQTime*

Always returned:

ChannelDisposition, *ChannelInstanceType*, *ChannelName*, *ChannelStatus*, *ChannelType*, *ConnectionName*, *RemoteApplTag*, *RemoteQMgrName*, *StopRequested*, *SubState*, *XmitQName*

Returned if requested:

Batches, *BatchSize*, *BatchSizeIndicator*, *BuffersReceived*, *BuffersSent*, *BytesReceived*, *BytesSent*, *ChannelMonitoring*, *ChannelStartDate*, *ChannelStartTime*, *ClientIdentifier*, *CompressionRate*, *CompressionTime*, *CurrentLUWID*, *CurrentMsgs*, *CurrentSequenceNumber*, *CurrentSharingConversations*, *ExitTime*, *HeaderCompression*, *HeartbeatInterval*, *InDoubtInbound*, *InDoubtStatus*, *InDoubtOutbound*, *KeepAliveInterval*, *LastLUWID*, *LastMsgDate*, *LastMsgTime*, *LastSequenceNumber*, *LocalAddress*, *LongRetriesLeft*, *MaxMsgLength*, *MaxSharingConversations*, *MCAJobName*, *MCAStatus*, *MCAUserIdentifier*, *MessageCompression*, *Msgs*, *MsgsAvailable*, *MsgsReceived*, *MsgsSent*, *NetTime*, *NonPersistentMsgSpeed*, *PendingOutbound*, *QMgrName*, *ResponseType*, *RemoteVersion*, *RemoteProduct*, *ShortRetriesLeft*, *SSLCertRemoteIssuerName*, *SSLCertUserId*, *SSLKeyResetDate*, *SSLKeyResets*, *SSLKeyResetTime*, *SSLShortPeerName*, *XQTime*

Response data

Batches **(MQCFIN)**
Number of completed batches (parameter identifier: MQIACH_BATCHES).

BatchSize **(MQCFIN)**
Negotiated batch size (parameter identifier: MQIACH_BATCH_SIZE).

BatchSizeIndicator **(MQCFIL)**
Indicator of the number of messages in a batch (parameter identifier: MQIACH_BATCH_SIZE_INDICATOR). Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

BuffersReceived **(MQCFIN)**
Number of buffers received (parameter identifier: MQIACH_BUFFERS_RCVD).

BuffersSent **(MQCFIN)**
Number of buffers sent (parameter identifier: MQIACH_BUFFERS_SENT).

BytesReceived **(MQCFIN)**
Number of bytes received (parameter identifier: MQIACH_BYTES_RCVD).

BytesSent **(MQCFIN)**
Number of bytes sent (parameter identifier: MQIACH_BYTES_SENT).

ChannelDisposition **(MQCFIN)**
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter is valid only on z/OS.

The value can be any of the following values:

MQCHLD_PRIVATE
Status information for a private channel.

MQCHLD_SHARED
Status information for a shared channel.

MQCHLD_FIXSHARED
Status information for a shared channel, tied to a specific queue manager.

ChannelInstanceType **(MQCFIN)**
Channel instance type (parameter identifier: MQIACH_CHANNEL_INSTANCE_TYPE).

The value can be:

MQOT_CURRENT_CHANNEL
Current channel status.

MQOT_SAVED_CHANNEL
Saved channel status.

MQOT_SHORT_CHANNEL
Short channel status, only on z/OS.

ChannelMonitoring **(MQCFIN)**
Current level of monitoring data collection for the channel (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_OFF
Monitoring for the channel is switched off.

MQMON_LOW
Low rate of data collection.

MQMON_MEDIUM
Medium rate of data collection.

MQMON_HIGH
High rate of data collection.

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStartDate **(MQCFST)**
Date channel started, in the form `yyyy-mm-dd` (parameter identifier: MQCACH_CHANNEL_START_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

ChannelStartTime **(MQCFST)**
Time channel started, in the form `hh.mm.ss` (parameter identifier: MQCACH_CHANNEL_START_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

ChannelStatus **(MQCFIN)**
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

Channel status has the following values defined:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_PAUSED
Channel is paused.

MQCHS_STOPPING
Channel is in process of stopping.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_STOPPED
Channel is stopped.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_SWITCHING
Channel is switching transmission queues.

MQCHS_INITIALIZING
Channel is initializing.

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

CompressionRate **(MQCFIL)**
The compression rate achieved displayed to the nearest percentage (parameter identifier: MQIACH_COMPRESSION_RATE). Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

CompressionTime **(MQCFIL)**
The amount of time per message, displayed in microseconds, spent during compression or decompression (parameter identifier: MQIACH_COMPRESSION_TIME). Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

ConnectionName **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_SHORT_CONN_NAME_LENGTH.

CurrentLUWID **(MQCFST)**
Logical unit of work identifier for in-doubt batch (parameter identifier: MQCACH_CURRENT_LUWID).

The logical unit of work identifier associated with the current batch, for a sending or a receiving channel.

For a sending channel, when the channel is in-doubt it is the LUWID of the in-doubt batch.

It is updated with the LUWID of the next batch when it is known.

The maximum length is MQ_LUWID_LENGTH.

CurrentMsgs **(MQCFIN)**
Number of messages in-doubt (parameter identifier: MQIACH_CURRENT_MSGS).

For a sending channel, this parameter is the number of messages that have been sent in the current batch. It is incremented as each message is sent, and when the channel becomes in-doubt it is the number of messages that are in-doubt.

For a receiving channel, it is the number of messages that have been received in the current batch. It is incremented as each message is received.

The value is reset to zero, for both sending and receiving channels, when the batch is committed.

CurrentSequenceNumber **(MQCFIN)**
Sequence number of last message in in-doubt batch (parameter identifier: MQIACH_CURRENT_SEQ_NUMBER).

For a sending channel, this parameter is the message sequence number of the last message sent. It is updated as each message is sent, and when the channel becomes in-doubt it is the message sequence number of the last message in the in-doubt batch.

For a receiving channel, it is the message sequence number of the last message that was received. It is updated as each message is received.

CurrentSharingConversations **(MQCFIN)**
Number of conversations currently active on this channel instance (parameter identifier: MQIACH_CURRENT_SHARING_CONVS).

This parameter is returned only for TCP/IP server-connection channels.

A value of zero indicates that the channel instance is running in a mode before IBM WebSphere MQ Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Client asynchronous consumption

ExitTime **(MQCFIL)**
Indicator of the time taken executing user exits per message (parameter identifier: MQIACH_EXIT_TIME_INDICATOR). Amount of time, in microseconds, spent processing user exits per message. Where more than one exit is executed per message, the value is the sum of all the user exit times for a single message. Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

HeaderCompression **(MQCFIL)**
Whether the header data sent by the channel is compressed (parameter identifier: MQIACH_HDR_COMPRESSION). Two values are returned:

- The default header data compression value negotiated for this channel.

- The header data compression value used for the last message sent. The header data compression value can be altered in a sending channels message exit. If no message has been sent, the second value is MQCOMPRESS_NOT_AVAILABLE.

The values can be:

MQCOMPRESS_NONE
No header data compression is performed. MQCOMPRESS_NONE is the default value.

MQCOMPRESS_SYSTEM
Header data compression is performed.

MQCOMPRESS_NOT_AVAILABLE
No message has been sent by the channel.

HeartbeatInterval **(MQCFIN)**
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

InDoubtStatus **(MQCFIN)**
Whether the channel is currently in doubt (parameter identifier: MQIACH_INDOUBT_STATUS).

A sending channel is only in doubt while the sending Message Channel Agent is waiting for an acknowledgment that a batch of messages, which it has sent, has been successfully received. It is not in doubt at all other times, including the period during which messages are being sent, but before an acknowledgment has been requested.

A receiving channel is never in doubt.

The value can be:

MQCHIDS_NOT_INDOUBT
Channel is not in-doubt.

MQCHIDS_INDOUBT
Channel is in-doubt.

KeepAliveInterval **(MQCFIN)**
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL). This parameter is valid only on z/OS.

LastLUWID **(MQCFST)**
Logical unit of work identifier for last committed batch (parameter identifier: MQCACH_LAST_LUWID).

The maximum length is MQ_LUWID_LENGTH.

LastMsgDate **(MQCFST)**
Date last message was sent, or MQI call was handled, in the form `yyyy-mm-dd` (parameter identifier: MQCACH_LAST_MSG_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

LastMsgTime **(MQCFST)**
Time last message was sent, or MQI call was handled, in the form `hh.mm.ss` (parameter identifier: MQCACH_LAST_MSG_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

LastSequenceNumber **(MQCFIN)**
Sequence number of last message in last committed batch (parameter identifier: MQIACH_LAST_SEQ_NUMBER).

LocalAddress **(MQCFST)**
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetriesLeft **(MQCFIN)**
Number of long retry attempts remaining (parameter identifier: MQIACH_LONG_RETRIES_LEFT).

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH). This parameter is valid only on z/OS.

MaxSharingConversations **(MQCFIN)**
Maximum number of conversations permitted on this channel instance. (parameter identifier: MQIACH_MAX_SHARING_CONVS)

This parameter is returned only for TCP/IP server-connection channels.

A value of zero indicates that the channel instance is running in a mode before IBM WebSphere MQ Version 7.0, regarding:

- Administrator stop-quiesce
- Heartbeating
- Read ahead
- Client asynchronous consumption

MCAJobName **(MQCFST)**
Name of MCA job (parameter identifier: MQCACH_MCA_JOB_NAME).

The maximum length of the string is MQ_MCA_JOB_NAME_LENGTH.

MCAStatus **(MQCFIN)**
MCA status (parameter identifier: MQIACH_MCA_STATUS).

The value can be:

MQMCAS_STOPPED
Message channel agent stopped.

MQMCAS_RUNNING
Message channel agent running.

MCAUserIdentifier **(MQCFST)**
The user ID used by the MCA (parameter identifier: MQCACH_MCA_USER_ID).

This parameter applies only to server-connection, receiver, requester, and cluster-receiver channels.

The maximum length of the string is MQ_MCA_USER_ID_LENGTH.

MessageCompression **(MQCFIL)**
Whether the header data sent by the channel is compressed (parameter identifier: MQIACH_MSG_COMPRESSION). Two values are returned:

- The default message data compression value negotiated for this channel.
- The message data compression value used for the last message sent. The message data compression value can be altered in a sending channels message exit. If no message has been sent, the second value is MQCOMPRESS_NOT_AVAILABLE.

The values can be:

MQCOMPRESS_NONE
No message data compression is performed. MQCOMPRESS_NONE is the default value.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_NOT_AVAILABLE
No message has been sent by the channel.

Msgs **(MQCFIN)**
Number of messages sent or received, or number of MQI calls handled (parameter identifier: MQIACH_MSGS).

MsgsAvailable **(MQCFIN)**
Number of messages available (parameter identifier: MQIACH_XMITQ_MSGS_AVAILABLE). Number of messages queued on the transmission queue available to the channel for MQGETs.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

This parameter applies to cluster sender channels only.

NetTime **(MQCFIL)**
Indicator of the time of a network operation (parameter identifier: MQIACH_NETWORK_TIME_INDICATOR). Amount of time, in microseconds, to send a request to the remote end of the channel and receive a response. This time only measures the network time for such an operation. Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

NonPersistentMsgSpeed **(MQCFIN)**
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

QMgrName **(MQCFST)**
Name of the queue manager that owns the channel instance (parameter identifier: MQCA_Q_MGR_NAME). This parameter is valid only on z/OS.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteApplTag **(MQCFST)**
The remote partner application name. This parameter is the name of the client application at the remote end of the channel. This parameter applies only to server-connection channels (parameter identifier: MQCACH_REMOTE_APPL_TAG).

RemoteProduct **(MQCFST)**
The remote partner product identifier. This parameter is the product identifier of the IBM WebSphere MQ code running at the remote end of the channel (parameter identifier: MQCACH_REMOTE_PRODUCT).

The possible values are shown in the following table:

Table 99. Product Identifier values

Product Identifier	Description
MQMM	Queue Manager (non z/OS Platform)
MQMV	Queue Manager on z/OS
MQCC	WebSphere MQ C client
MQNM	WebSphere MQ .NET fully managed client
MQJB	WebSphere MQ Classes for JAVA
MQJM	WebSphere MQ Classes for JMS (normal mode)
MQJN	WebSphere MQ Classes for JMS (migration mode)
MQJU	Common Java interface to the MQI
MQXC	XMS client C/C++ (normal mode)
MQXD	XMS client C/C++ (migration mode)
MQXN	XMS client .NET (normal mode)
MQXM	XMS client .NET (migration mode)
MQXU	WebSphere MQ .NET XMS client (unmanaged/XA)
MQNU	WebSphere MQ .NET unmanaged client

RemoteVersion **(MQCFST)**
The remote partner version. This parameter is the version of the IBM WebSphere MQ code running at the remote end of the channel (parameter identifier: MQCACH_REMOTE_VERSION).

The remote version is displayed as **VVRRMMFF**, where

VV Version

RR Release

MM Maintenance level

FF Fix level

RemoteQMgrName **(MQCFST)**
Name of the remote queue manager, or queue-sharing group (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

ShortRetriesLeft **(MQCFIN)**
Number of short retry attempts remaining (parameter identifier: MQIACH_SHORT_RETRIES_LEFT).

SSLCertRemoteIssuerName **(MQCFST)**
The full Distinguished Name of the issuer of the remote certificate. The issuer is the certificate authority that issued the certificate (parameter identifier: MQCACH_SSL_CERT_ISSUER_NAME).

The maximum length of the string is MQ_SHORT_DNAME_LENGTH.

SSLCertUserId **(MQCFST)**
The local user ID associated with the remote certificate (parameter identifier: MQCACH_SSL_CERT_USER_ID).

This parameter is valid only on z/OS.

The maximum length of the string is MQ_USER_ID_LENGTH.

SSLKeyResetDate **(MQCFST)**
Date of the previous successful SSL secret key reset, in the form `yyyy-mm-dd` (parameter identifier: MQCACH_SSL_KEY_RESET_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

SSLKeyResets **(MQCFIN)**
SSL secret key resets (parameter identifier: MQIACH_SSL_KEY_RESETS).

The number of successful SSL secret key resets that have occurred for this channel instance since the channel started. If SSL secret key negotiation is enabled, the count is incremented whenever a secret key reset is performed.

SSLKeyResetTime **(MQCFST)**
Time of the previous successful SSL secret key reset, in the form `hh.mm.ss` (parameter identifier: MQCACH_SSL_KEY_RESET_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

SSLShortPeerName **(MQCFST)**
Distinguished Name of the peer queue manager or client at the other end of the channel (parameter identifier: MQCACH_SSL_SHORT_PEER_NAME).

The maximum length is MQ_SHORT_DNAME_LENGTH. This limit might mean that exceptionally long Distinguished Names are truncated.

StopRequested **(MQCFIN)**
Whether user stop request is outstanding (parameter identifier: MQIACH_STOP_REQUESTED).

The value can be:

MQCHSR_STOP_NOT_REQUESTED
User stop request has not been received.

MQCHSR_STOP_REQUESTED
User stop request has been received.

SubState **(MQCFIN)**
Current action being performed by the channel (parameter identifier: MQIACH_CHANNEL_SUBSTATE).

The value can be:

MQCHSSTATE_CHADEXIT
Running channel auto-definition exit.

MQCHSSTATE_COMPRESSING
Compressing or decompressing data.

MQCHSSTATE_END_OF_BATCH
End of batch processing.

MQCHSSTATE_HANDSHAKING
SSL handshaking.

MQCHSSTATE_HEARTBEATING
Heartbeating with partner.

MQCHSSTATE_IN_MQGET
Performing MQGET.

MQCHSSTATE_IN_MQI_CALL
Executing an WebSphere MQ API call, other than an MQPUT or MQGET.

MQCHSSTATE_IN_MQPUT
Performing MQPUT.

MQCHSSTATE_MREXIT
Running retry exit.

MQCHSSTATE_MSGEXIT
Running message exit.

MQCHSSTATE_NAME_SERVER
Name server request.

MQCHSSTATE_NET_CONNECTING
Network connect.

MQCHSSTATE_OTHER
Undefined state.

MQCHSSTATE_RCVEXIT
Running receive exit.

MQCHSSTATE_RECEIVING
Network receive.

MQCHSSTATE_RESYNCHING
Resynching with partner.

MQCHSSTATE_SCYEXIT
Running security exit.

MQCHSSTATE_SENDEXIT
Running send exit.

MQCHSSTATE_SENDING
Network send.

MQCHSSTATE_SERIALIZING
Serialized on queue manager access.

XmitQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XQTime **(MQCFIL)**
Transmission queue time indicator (parameter identifier: MQIACH_XMITQ_TIME_INDICATOR). The time, in microseconds, that messages remained on the transmission queue before being retrieved. The time is measured from when the message is put onto the transmission queue until it is retrieved to be sent on the channel and, therefore, includes any interval caused by a delay in the putting application.

Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned.

Inquire Channel Status (Response):

The response to the Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command consists of the response header followed by the *ChannelName* structure and the requested combination of attribute parameter structures.

One such message is generated for each channel instance found that matches the criteria specified on the command.

Always returned:
ChannelName, *ChannelStatus*, *ChannelType*

Returned if requested:
ChannelStartDate, *ChannelStartTime*, *ClientIdentifier*, *ConnectionName*, *InDoubtInbound*, *InDoubtOutbound*, *KeepAliveInterval*, *LastMsgTime*, *MCAUserIdentifier*, *MsgsReceived*, *MsgsSent*, *PendingOutbound*, *ResponseType*

Response data

ChannelStartDate **(MQCFST)**
Date channel started, in the form yyyy-mm-dd (parameter identifier: MQCACH_CHANNEL_START_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

ChannelStartTime **(MQCFST)**
Time channel started, in the form hh.mm.ss (parameter identifier: MQCACH_CHANNEL_START_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

ChannelStatus **(MQCFIN)**
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be:

MQCHS_DISCONNECTED
Channel is disconnected.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value must be:

MQCHT_MQTT
Telemetry.

ClientIdentifier **(MQCFST)**
The ClientID of the client (parameter identifier: MQCACH_CLIENT_ID).

The maximum length of the string is MQ_CLIENT_ID_LENGTH.

ConnectionName **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

InDoubtInBound **(MQCFIN)**
The number of inbound messages to the client that are in doubt (parameter identifier: MQIACH_IN_DOUBT_IN).

InDoubtoutBound **(MQCFIN)**
The number of outbound messages from the client that are in doubt (parameter identifier: MQIACH_IN_DOUBT_OUT).

KeepAliveInterval **(MQCFIN)**
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

The interval in milliseconds after which the client is disconnected because of inactivity. If the telemetry (MQXR) service does not receive any communication from the client within the keep alive interval, it disconnects from the client. This interval is calculated based on the MQTT keep alive time sent by the client when it connects. The maximum size is MQ_MQTT_MAX_KEEP_ALIVE.

LastMsgTime **(MQCFST)**
Time last message was sent, or MQI call was handled, in the form hh.mm.ss (parameter identifier: MQCACH_LAST_MSG_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

MsgsReceived **(MQCFIN64)**
Number of messages received by the client since it last connected (parameter identifier: MQIACH_MSGS_RECEIVED / MQIACH_MSGS_RCVD).

MsgsSent **(MQCFIN64)**
Number of messages sent by the client since it last connected (parameter identifier: MQIACH_MSGS_SENT).

PendingOutbound **(MQCFIN)**
The number of outbound messages pending (parameter identifier: MQIACH_PENDING_OUT).

ResponseType **(MQCFIL)**
Response type (parameter identifier: MQIACF_RESPONSE_TYPE). This parameter is for MQTT channels only.

This MQTT channel parameter specifies the type of response that is required. The type of response is based on the one of the following three values:

- If **ResponseType** is set to MQRESP_NORMAL or if it is not specified, the following structures are returned:
  - The **ChannelName** structure.
  - The **ClientIdentifer** structure.
  - The **ChannelType** structure.

  All the remaining 'usual' structures and requested structures are returned as normal.
- If **ResponseType** is set to MQRESP_SUMMARY, the following structures are returned:
  - The **ChannelName** structure.
  - The **ChannelType** structure.

  the **ConversationCount** structure is also returned if it was requested.
- If **ResponseType** is set to MQRESP_TOTAL, only the **ConversationCount** structure is returned if it was requested.

Inquire Cluster Queue Manager:

The Inquire Cluster Queue Manager (MQCMD_INQUIRE_CLUSTER_Q_MGR) command inquires about the attributes of WebSphere MQ queue managers in a cluster.

HP Integrity NonStop Server	UNIX and Linux	Windows
✓	✓	✓

Required parameters

ClusterQMgrName **(MQCFST)**
Queue manager name (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

Generic queue manager names are supported. A generic name is a character string followed by an asterisk "*", for example ABC*. It selects all queue managers having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue manager name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Optional parameters

Channel **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies that eligible cluster queue managers are limited to those having the specified channel name.

Generic channel names are supported. A generic name is a character string followed by an asterisk "*", for example ABC*. It selects all queue managers having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

If you do not specify a value for this parameter, channel information about *all* queue managers in the cluster is returned.

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

Specifies that eligible cluster queue managers are limited to those having the specified cluster name.

Generic cluster names are supported. A generic name is a character string followed by an asterisk "*", for example ABC*. It selects all queue managers having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

If you do not specify a value for this parameter, cluster information about *all* queue managers inquired is returned.

ClusterQMgrAttrs **(MQCFIL)**
Attributes (parameter identifier: MQIACF_CLUSTER_Q_MGR_ATTRS).

Some parameters are relevant only for cluster channels of a particular type or types. Attributes that are not relevant for a particular type of channel cause no output, and do not cause an error. To check which attributes apply to which channel types; see Channel attributes and channel types.

The attribute list might specify the following value on its own. If the parameter is not specified, a default value is used.

MQIACF_ALL
All attributes.

Alternative, supply a combination of the following values:

MQCA_ALTERATION_DATE
The date on which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQCA_CLUSTER_DATE
The date on which the information became available to the local queue manager.

MQCA_CLUSTER_NAME
The name of the cluster to which the channel belongs.

MQCA_CLUSTER_Q_MGR_NAME
The name of the cluster to which the channel belongs.

MQCA_CLUSTER_TIME
The time at which the information became available to the local queue manager.

MQCA_Q_MGR_IDENTIFIER
The unique identifier of the queue manager.

MQCA_XMIT_Q_NAME
The cluster transmission queue used by the queue manager. The property is only available on platforms other than z/OS.

MQCACH_CONNECTION_NAME
Connection name.

MQCACH_DESCRIPTION
Description.

MQCACH_LOCAL_ADDRESS
Local communications address for the channel.

MQCACH_MCA_NAME
Message channel agent name.

You cannot use MQCACH_MCA_NAME as a parameter to filter on.

MQCACH_MCA_USER_ID
MCA user identifier.

MQCACH_MODE_NAME
Mode name.

MQCACH_MR_EXIT_NAME
Message-retry exit name.

MQCACH_MR_EXIT_USER_DATA
Message-retry exit user data.

MQCACH_MSG_EXIT_NAME
Message exit name.

MQCACH_MSG_EXIT_USER_DATA
Message exit user data.

MQCACH_PASSWORD
Password.

This parameter is not valid on z/OS.

MQCACH_RCV_EXIT_NAME
Receive exit name.

MQCACH_RCV_EXIT_USER_DATA
Receive exit user data.

MQCACH_SEC_EXIT_NAME
Security exit name.

MQCACH_SEC_EXIT_USER_DATA
Security exit user data.

MQCACH_SEND_EXIT_NAME
Send exit name.

MQCACH_SEND_EXIT_USER_DATA
Send exit user data.

MQCACH_SSL_CIPHER_SPEC
SSL cipher spec.

MQIACH_SSL_CLIENT_AUTH
SSL client authentication.

MQCACH_SSL_PEER_NAME
SSL peer name.

MQCACH_TP_NAME
Transaction program name.

MQCACH_USER_ID
User identifier.

This parameter is not valid on z/OS.

MQIA_MONITORING_CHANNEL
Online monitoring data collection.

MQIA_USE_DEAD_LETTER_Q
Determines whether the dead-letter queue is used when messages cannot be delivered by channels.

MQIACF_Q_MGR_DEFINITION_TYPE
How the cluster queue manager was defined.

MQIACF_Q_MGR_TYPE
The function of the queue manager in the cluster.

MQIACF_SUSPEND
Specifies whether the queue manager is suspended from the cluster.

MQIACH_BATCH_HB
The value being used for the batch heartbeat.

MQIACH_BATCH_INTERVAL
Batch wait interval (seconds).

MQIACH_BATCH_DATA_LIMIT
Batch data limit (kilobytes).

MQIACH_BATCH_SIZE
Batch size.

MQIACH_CHANNEL_STATUS
Channel status.

MQIACH_CLWL_CHANNEL_PRIORITY
Cluster workload channel priority.

MQIACH_CLWL_CHANNEL_RANK
Cluster workload channel rank.

MQIACH_CLWL_CHANNEL_WEIGHT
Cluster workload channel weight.

MQIACH_DATA_CONVERSION
Specifies whether sender must convert application data.

MQIACH_DISC_INTERVAL
Disconnection interval.

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

MQIACH_HDR_COMPRESSION
The list of header data compression techniques supported by the channel.

MQIACH_KEEP_ALIVE_INTERVAL
KeepAlive interval (valid on z/OS only).

MQIACH_LONG_RETRY
Count of long duration attempts.

MQIACH_LONG_TIMER
Long duration timer.

MQIACH_MAX_MSG_LENGTH
Maximum message length.

MQIACH_MCA_TYPE
MCA type.

MQIACH_MR_COUNT
Count of send message attempts.

MQIACH_MR_INTERVAL
Interval between attempting to resend a message in milliseconds.

MQIACH_MSG_COMPRESSION
List of message data compression techniques supported by the channel.

MQIACH_NETWORK_PRIORITY
Network priority.

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

MQIACH_PUT_AUTHORITY
Put authority.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQIACH_SHORT_RETRY
Count of short duration attempts.

MQIACH_SHORT_TIMER
Short duration timer.

MQIACH_XMIT_PROTOCOL_TYPE
Transmission protocol type.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk "*". The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ClusterQMgrAttrs* except MQIACF_ALL and others as noted. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ClusterQMgrAttrs* except MQCA_CLUSTER_Q_MGR_NAME and others as noted. Use this parameter to

restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter for *Channel* or *ClusterName*, you cannot also specify the *Channel* or *ClusterName* parameter.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Inquire Cluster Queue Manager (Response):

The response to the Inquire Cluster Queue Manager (MQCMD_INQUIRE_CLUSTER_Q_MGR) command consists of three parts. The response header is followed by the *QMgrName* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Always returned:
ChannelName, *ClusterName*, *QMgrName*,

Returned if requested:
AlterationDate, *AlterationTime*, *BatchHeartbeat*, *BatchInterval*, *BatchSize*, *ChannelDesc*, *ChannelMonitoring*, *ChannelStatus*, *ClusterDate*, *ClusterInfo*, *ClusterTime*, *CLWLChannelPriority*, *CLWLChannelRank*, *CLWLChannelWeight*, *ConnectionName*, *DataConversion*, *DiscInterval*, *HeaderCompression*, *HeartbeatInterval*, *KeepAliveInterval*, *LocalAddress*, *LongRetryCount*, *LongRetryInterval*, *MaxMsgLength*, *MCAName*, *MCAType*, *MCAUserIdentifier*,

MessageCompression, *ModeName*, *MsgExit*, *MsgRetryCount*, *MsgRetryExit*, *MsgRetryInterval*, *MsgRetryUserData*, *MsgUserData*, *NetworkPriority*, *NonPersistentMsgSpeed*, *Password*, *PutAuthority*, *QMgrDefinitionType*, *QMgrIdentifier*, *QMgrType*, *ReceiveExit*, *ReceiveUserData*, *SecurityExit*, *SecurityUserData*, *SendExit*, *SendUserData*, *SeqNumberWrap*, *ShortRetryCount*, *ShortRetryInterval*, *SSLCipherSpec*, *SSLClientAuth*, *SSLPeerName*, *Suspend*, *TpName* , *TransmissionQName*, *TransportType*, *UseDLQ*, *UserIdentifier*

Response data

AlterationDate **(MQCFST)**
Alteration date, in the form yyyy-mm-dd (parameter identifier: MQCA_ALTERATION_DATE).

The date at which the information was last altered.

AlterationTime **(MQCFST)**
Alteration time, in the form hh.mm.ss (parameter identifier: MQCA_ALTERATION_TIME).

The time at which the information was last altered.

BatchHeartbeat **(MQCFIN)**
The value being used for the batch heartbeat (parameter identifier: MQIACH_BATCH_HB).

The value can be 0 - 999,999. A value of 0 indicates that the batch heartbeat is not being used.

BatchInterval **(MQCFIN)**
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

BatchSize **(MQCFIN)**
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

ChannelDesc **(MQCFST)**
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

ChannelMonitoring **(MQCFIN)**
Online monitoring data collection (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_OFF
Online monitoring data collection is turned off for this channel.

MQMON_Q_MGR
The value of the queue manager's *ChannelMonitoring* parameter is inherited by the channel. MQMON_Q_MGR is the default value.

MQMON_LOW
Online monitoring data collection is turned on, with a low rate of data collection, for this channel unless the queue manager's *ChannelMonitoring* parameter is MQMON_NONE.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate rate of data collection, for this channel unless the queue manager's *ChannelMonitoring* parameter is MQMON_NONE.

MQMON_HIGH
Online monitoring data collection is turned on, with a high rate of data collection, for this channel unless the queue manager's *ChannelMonitoring* parameter is MQMON_NONE.

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelStatus **(MQCFIN)**
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_INACTIVE
Channel is not active.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_PAUSED
Channel is paused.

MQCHS_STOPPING
Channel is in process of stopping.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_STOPPED
Channel is stopped.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_INITIALIZING
Channel is initializing.

This parameter is returned if the channel is a cluster-sender channel (CLUSSDR) only.

ClusterDate **(MQCFST)**
Cluster date, in the form yyyy-mm-dd (parameter identifier: MQCA_CLUSTER_DATE).

The date at which the information became available to the local queue manager.

ClusterInfo **(MQCFIN)**
Cluster information (parameter identifier: MQIACF_CLUSTER_INFO).

The cluster information available to the local queue manager.

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterTime **(MQCFST)**
Cluster time, in the form hh.mm.ss (parameter identifier: MQCA_CLUSTER_TIME).

The time at which the information became available to the local queue manager.

CLWLChannelPriority **(MQCFIN)**
Channel priority (parameter identifier: MQIACH_CLWL_CHANNEL_PRIORITY).

CLWLChannelRank **(MQCFIN)**
Channel rank (parameter identifier: MQIACH_CLWL_CHANNEL_RANK).

CLWLChannelWeight **(MQCFIN)**
Channel weighting (parameter identifier: MQIACH_CLWL_CHANNEL_WEIGHT).

ConnectionName **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH. On z/OS, it is MQ_LOCAL_ADDRESS_LENGTH.

DataConversion **(MQCFIN)**
Specifies whether sender must convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DiscInterval **(MQCFIN)**
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

HeaderCompression **(MQCFIL)**
Header data compression techniques supported by the channel (parameter identifier: MQIACH_HDR_COMPRESSION). The values specified are in order of preference.

The value can be one, or more, of

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval **(MQCFIN)**
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

KeepAliveInterval **(MQCFIN)**
KeepAlive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL). This parameter applies to z/OS only.

LocalAddress **(MQCFST)**
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetryCount **(MQCFIN)**
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

LongRetryInterval **(MQCFIN)**
Long timer (parameter identifier: MQIACH_LONG_TIMER).

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

MCAName **(MQCFST)**
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

The maximum length of the string is MQ_MCA_NAME_LENGTH.

MCAType **(MQCFIN)**
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

The value can be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (Windows only).

MCAUserIdentifier **(MQCFST)**
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

MessageCompression **(MQCFIL)**
Message data compression techniques supported by the channel (parameter identifier: MQIACH_MSG_COMPRESSION). The values specified are in order of preference.

The value can be one, or more, of:

MQCOMPRESS_NONE
No message data compression is performed.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

ModeName **(MQCFST)**
Mode name (parameter identifier: MQCACH_MODE_NAME).

The maximum length of the string is MQ_MODE_NAME_LENGTH.

MsgExit **(MQCFST)**
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

In the following environments more than one message exit can be defined for a channel. If more than one message exit is defined, the list of names is returned in an MQCFSL structure instead of an (MQCFST) structure. The environments are: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. An MQCFSL structure is always used on z/OS.

MsgRetryCount **(MQCFIN)**
Message retry count (parameter identifier: MQIACH_MR_COUNT).

MsgRetryExit **(MQCFST)**
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

MsgRetryInterval **(MQCFIN)**
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

MsgRetryUserData **(MQCFST)**
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgUserData **(MQCFST)**
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, more than one message exit user data string can be defined for a channel. If more than one string is defined, the list of strings is returned in an MQCFSL structure instead of an (MQCFST) structure. The environments are: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. An MQCFSL structure is always used on z/OS.

NetworkPriority **(MQCFIN)**
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

NonPersistentMsgSpeed **(MQCFIN)**
Speed at which non-persistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password **(MQCFST)**
Password (parameter identifier: MQCACH_PASSWORD). This parameter is not available on z/OS.

If a nonblank password is defined, it is returned as asterisks. Otherwise, it is returned as blanks.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

PutAuthority **(MQCFIN)**
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

The value can be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

MQPA_ALTERNATE_OR_MCA
The user identifier from the *UserIdentifier* field of the message descriptor is used. Any user ID received from the network is not used. This value is valid only on z/OS.

MQPA_ONLY_MCA
The default user identifier is used. Any user ID received from the network is not used. This value is valid only on z/OS.

QMgrDefinitionType **(MQCFIN)**
Queue manager definition type (parameter identifier: MQIACF_Q_MGR_DEFINITION_TYPE).

The value can be:

MQQMDT_EXPLICIT_CLUSTER_SENDER
A cluster-sender channel from an explicit definition.

MQQMDT_AUTO_CLUSTER_SENDER
A cluster-sender channel by auto-definition.

MQQMDT_CLUSTER_RECEIVER
A cluster-receiver channel.

MQQMDT_AUTO_EXP_CLUSTER_SENDER
A cluster-sender channel, both from an explicit definition and by auto-definition.

QMgrIdentifier **(MQCFST)**
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

QMgrName **(MQCFST)**
Queue manager name (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QMgrType **(MQCFIN)**
Queue manager type (parameter identifier: MQIACF_Q_MGR_TYPE).

The value can be:

MQQMT_NORMAL
A normal queue manager.

MQQMT_REPOSITORY
A repository queue manager.

ReceiveExit **(MQCFST)**
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

In the following environments, more than one receive exit can be defined for a channel. If more than one receive exit is defined, the list of names is returned in an MQCFSL structure instead of an (MQCFST) structure. The environments are: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. An MQCFSL structure is always used on z/OS.

ReceiveUserData **(MQCFST)**
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, more than one receive exit user data string can be defined for the channel. If more than one string is defined, the list of strings is returned in an MQCFSL structure instead of an (MQCFST) structure. The environments are: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. An MQCFSL structure is always used on z/OS.

SecurityExit **(MQCFST)**
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

SecurityUserData **(MQCFST)**
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit **(MQCFST)**
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

In the following environments, more than one send exit can be defined for a channel. If more than one send exit is defined, the list of names is returned in an MQCFSL structure instead of an (MQCFST) structure. The environments are: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. An MQCFSL structure is always used on z/OS.

SendUserData **(MQCFST)**
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, more than one send exit user data string can be defined for the channel. If more than one string is defined, the list of strings is returned in an MQCFSL structure instead of an (MQCFST) structure. The environments are: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. An MQCFSL structure is always used on z/OS.

SeqNumberWrap **(MQCFIN)**
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

ShortRetryCount **(MQCFIN)**
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

ShortRetryInterval **(MQCFIN)**
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

SSLCipherSpec **(MQCFST)**
CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

SSLClientAuth **(MQCFIN)**
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Client authentication required

MQSCA_OPTIONAL
Client authentication is optional.

Defines whether WebSphere MQ requires a certificate from the SSL client.

SSLPeerName **(MQCFST)**
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The length of the string is MQ_SSL_PEER_NAME_LENGTH. On z/OS, it is MQ_SHORT_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the distinguished name of the certificate from the peer queue manager or client at the other end of the channel. (A distinguished name is the identifier of the SSL certificate.) If the distinguished name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

Suspend **(MQCFIN)**
Specifies whether the queue manager is suspended (parameter identifier: MQIACF_SUSPEND).

The value can be:

MQSUS_NO
The queue manager is not suspended from the cluster.

MQSUS_YES
The queue manager is suspended from the cluster.

TpName **(MQCFST)**
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The maximum length of the string is MQ_TP_NAME_LENGTH.

TranmissionQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME). The cluster transmission queue used by the queue manager. The property is only available on platforms other than z/OS.

The maximum length of the string is MQ_Q_NAME_LENGTH.

TransportType **(MQCFIN)**
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

MQXPT_DECNET
DECnet.

UseDLQ **(MQCFIN)**
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

UserIdentifier **(MQCFST)**
Task user identifier (parameter identifier: MQCACH_USER_ID). This parameter is not available on z/OS.

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

Inquire Communication Information Object:

The Inquire Communication Information Object (MQCMD_INQUIRE_COMM_INFO) command inquires about the attributes of existing WebSphere MQ communication information objects.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters:
CominfoName

Optional parameters:
CominfoAttrs, *IntegerFilterCommand*, *StringFilterCommand*

Required parameters

CominfoName **(MQCFST)**
The name of the communication information definition about which information is to be returned (parameter identifier: MQCA_COMM_INFO_NAME).

The communication information name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Optional parameters

CominfoAttrs **(MQCFIL)**
Cominfo attributes (parameter identifier: MQIACF_COMM_INFO_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQIA_CODED_CHAR_SET_ID
CCSID for transmitted messages.

MQIA_COMM_EVENT
Cominfo event control.

MQIA_MCAST_BRIDGE
Multicast bridging.

MQIA_MONITOR_INTERVAL
Frequency of update for monitoring information.

MQIACF_ENCODING
Encoding for transmitted messages.

MQIACH_MC_HB_INTERVAL
Multicast heartbeat interval.

MQIACH_MSG_HISTORY
Amount of message history being kept.

MQIACH_MULTICAST_PROPERTIES
Multicast properties control.

MQIACH_NEW_SUBSCRIBER_HISTORY
New subscriber history.

MQIACH_PORT
Port Number.

MQCA_ALTERATION_DATE
The date on which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQCA_COMM_INFO_DESC
Cominfo description.

MQCA_COMM_INFO_TYPE
Cominfo type

MQCACH_GROUP_ADDRESS
Group Address.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *CominfoAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter for *CominfoType* (MQIA_COMM_INFO_TYPE), you cannot also specify the *CominfoType* parameter.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *CominfoAttrs* except MQCA_COMM_INFO_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Inquire Communication Information Object (Response):

The response to the Inquire Communication Information Object (MQCMD_INQUIRE_COMM_INFO) command consists of the response header followed by the CominfoName structure, and the requested combination of attribute parameter structures (where applicable).

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

If a generic communication information name was specified, one such message is generated for each object found.

Always returned:
CominfoName

Returned if requested:
AlterationDate, *AlterationTime*, *Bridge*, *CCSID*, *CommEvent*, *Description*, *Encoding*, *GrpAddress*, *MonitorInterval*, *MulticastHeartbeat*, *MulticastPropControl*, *MsgHistory*, *NewSubHistory*, *PortNumber*, *Type*

Response data

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss`.

Bridge **(MQCFIN)**
Multicast Bridging (parameter identifier: MQIA_MCAST_BRIDGE).

Controls whether publications from applications not using Multicast are bridged to applications using multicast.

CCSID **(MQCFIN)**
CCSID that messages are trasmitted in (parameter identifier: MQIA_CODED_CHAR_SET_ID).

The coded character set identifier that messages are transmitted in.

CommEvent **(MQCFIN)**
Event Control (parameter identifier: MQIA_COMM_EVENT).

Controls whether event messages are generated for multicast handles that are created using this COMMINFO object. The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_EXCEPTION
Reporting of events for message reliability below the reliability threshold enabled.

CominfoName **(MQCFST)**
The name of the communication information definition (parameter identifier: MQCA_COMM_INFO_NAME).

The maximum length of the string is MQ_COMM_INFO_NAME_LENGTH.

Description **(MQCFST)**
Description of the communication information definition (parameter identifier: MQCA_COMM_INFO_DESC).

The maximum length of the string is MQ_COMM_INFO_DESC_LENGTH.

Encoding **(MQCFIN)**
Encoding that messages are transmitted in (parameter identifier: MQIACF_ENCODING).

The encoding that messages are transmitted in. The value can be:

MQENC_AS_PUBLISHED
Encoding taken from published message.

MQENC_NORMAL

MQENC_REVERSED

MQENC_S390

MQENC_TNS

GrpAddress **(MQCFST)**
The group IP address or DNS name (parameter identifier: MQCACH_GROUP_ADDRESS).

The maximum length of the string is MQ_GROUP_ADDRESS_LENGTH.

MonitorInterval **(MQCFIN)**
Frequency of monitoring (parameter identifier: MQIA_MONITOR_INTERVAL).

How frequently, in seconds, monitoring information is updated and event messages are generated.

MulticastHeartbeat **(MQCFIN)**
Heartbeat Interval for multicast (parameter identifier: MQIACH_MC_HB_INTERVAL).

The heartbeat interval, in milliseconds, for multicast transmitters.

MulticastPropControl **(MQCFIN)**
Multicast property control (parameter identifier: MQIACH_MULTICAST_PROPERTIES).

Control which MQMD properties and user properties flow with the message. The value can be:

MQMCP_ALL
All MQMD and user properties.

MQMAP_REPLY
Properties related to replying to messages.

MQMAP_USER
Only user properties.

MQMAP_NONE
No MQMD or user properties.

MQMAP_COMPAT
Properties are transmitted in a format compatible with previous Multicast clients.

MsgHistory **(MQCFIN)**
Message History (parameter identifier: MQIACH_MSG_HISTORY).

The amount of message history, in kilobytes, that is kept by the system to handle retransmissions in the case of NACKS.

NewSubHistory **(MQCFIN)**
New Subscriber History (parameter identifier: MQIACH_NEW_SUBSCRIBER_HISTORY).

Controls how much historical data a new subscriber receives. The value can be:

MQNSH_NONE
Only publications from the time of the subscription are sent.

MQNSH_ALL
As much history as is known is retransmitted.

PortNumber **(MQCFIN)**
Port Number (parameter identifier: MQIACH_PORT).

The port number to transmit on.

Type **(MQCFIN)**
The type of the communications information definition (parameter identifier: MQIA_COMM_INFO_TYPE).

The value can be:

MQCIT_MULTICAST
Multicast.

Inquire Connection:

The Inquire connection (MQCMD_INQUIRE_CONNECTION) command inquires about the applications which are connected to the queue manager, the status of any transactions that those applications are running, and the objects which the application has open.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ConnectionId **(MQCFBS)**
Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

This parameter is the unique connection identifier associated with an application that is connected to the queue manager. Specify either this parameter **or** *GenericConnectionId*.

All connections are assigned a unique identifier by the queue manager regardless of how the connection is established.

If you need to specify a generic connection identifier, use the *GenericConnectionId* parameter instead.

The length of the string is MQ_CONNECTION_ID_LENGTH.

GenericConnectionId **(MQCFBS)**
Generic specification of a connection identifier (parameter identifier: MQBACF_GENERIC_CONNECTION_ID).

Specify either this parameter **or** *ConnectionId*.

If you specify a byte string of zero length, or one which contains only null bytes, information about all connection identifiers is returned. This value is the only value permitted for *GenericConnectionId*.

The length of the string is MQ_CONNECTION_ID_LENGTH.

Optional parameters

ByteStringFilterCommand **(MQCFBF)**
Byte string filter command descriptor. The parameter identifier must be MQBACF_EXTERNAL_UOW_ID, MQBACF_ORIGIN_UOW_ID, or MQBACF_Q_MGR_UOW_ID. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFBF - PCF byte string filter parameter" on page 1219 for information about using this filter condition.

If you specify a byte string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter, or a string filter using the *StringFilterCommand* parameter.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_Q_MGR_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

ConnectionAttrs **(MQCFIL)**
Connection attributes (parameter identifier: MQIACF_CONNECTION_ATTRS).

The attribute list can specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes of the selected *ConnInfoType*.

or, if you select a value of MQIACF_CONN_INFO_CONN for *ConnInfoType*, a combination of the following:

MQBACF_CONNECTION_ID
Connection identifier.

MQBACF_EXTERNAL_UOW_ID
External unit of recovery identifier associated with the connection.

MQBACF_ORIGIN_UOW_ID
Unit of recovery identifier assigned by the originator (valid on z/OS only).

MQBACF_Q_MGR_UOW_ID
Unit of recovery identifier assigned by the queue manager.

MQCACF_APPL_TAG
Name of an application that is connected to the queue manager.

MQCACF_ASID
The 4-character address-space identifier of the application identified in MQCACF_APPL_TAG (valid on z/OS only).

MQCACF_ORIGIN_NAME
Originator of the unit of recovery (valid on z/OS only).

MQCACF_PSB_NAME
The 8-character name of the program specification block (PSB) associated with the running IMS transaction (valid on z/OS only).

MQCACF_PST_ID
The 4-character IMS program specification table (PST) region identifier for the connected IMS region (valid on z/OS only).

MQCACF_TASK_NUMBER
A 7-digit CICS task number (valid on z/OS only).

MQCACF_TRANSACTION_ID
A 4-character CICS transaction identifier (valid on z/OS only).

MQCACF_UOW_LOG_EXTENT_NAME
Name of the first extent required to recover the transaction.
MQCACF_UOW_LOG_EXTENT_NAME is not valid on z/OS.

MQCACF_UOW_LOG_START_DATE
Date on which the transaction associated with the current connection first wrote to the log.

MQCACF_UOW_LOG_START_TIME
Time at which the transaction associated with the current connection first wrote to the log.

MQCACF_UOW_START_DATE
Date on which the transaction associated with the current connection was started.

MQCACF_UOW_START_TIME
Time at which the transaction associated with the current connection was started.

MQCACF_USER_IDENTIFIER
User identifier of the application that is connected to the queue manager.

MQCACH_CHANNEL_NAME
Name of the channel associated with the connected application.

MQCACH_CONNECTION_NAME
Connection name of the channel associated with the application.

MQIA_APPL_TYPE
Type of the application that is connected to the queue manager.

MQIACF_CONNECT_OPTIONS
Connect options currently in force for this application connection.

You cannot use the value MQCNO_STANDARD_BINDING as a filter value.

MQIACF_PROCESS_ID
Process identifier of the application that is currently connected to the queue manager.

This parameter is not valid on z/OS.

MQIACF_THREAD_ID
Thread identifier of the application that is currently connected to the queue manager.

This parameter is not valid on z/OS.

MQIACF_UOW_STATE
State of the unit of work.

MQIACF_UOW_TYPE
Type of external unit of recovery identifier as understood by the queue manager.

or, if you select a value of MQIACF_CONN_INFO_HANDLE for *ConnInfoType*, a combination of the following:

MQCACF_OBJECT_NAME
Name of each object that the connection has open.

MQCACH_CONNECTION_NAME
Connection name of the channel associated with the application.

MQIA_QSG_DISP
Disposition of the object (valid on z/OS only).

You cannot use MQIA_QSG_DISP as a parameter to filter on.

MQIA_READ_AHEAD
The read ahead connection status.

MQIA_UR_DISP
The unit of recovery disposition associated with the connection (valid on z/OS only).

MQIACF_HANDLE_STATE
Whether an API call is in progress.

MQIACF_OBJECT_TYPE
Type of each object that the connection has open.

MQIACF_OPEN_OPTIONS
Options used by the connection to open each object.

or, if you select a value of MQIACF_CONN_INFO_ALL for *ConnInfoType*, any of the previous values.

ConnInfoType **(MQCFIN)**
Type of connection information to be returned (parameter identifier: MQIACF_CONN_INFO_TYPE).

The value can be:

MQIACF_CONN_INFO_CONN
Connection information. On z/OS, MQIACF_CONN_INFO_CONN includes threads which might be logically or actually disassociated from a connection, together with those threads that are in-doubt and for which external intervention is needed to resolve them. MQIACF_CONN_INFO_CONN is the default value used if the parameter is not specified.

MQIACF_CONN_INFO_HANDLE
Information pertaining only to those objects opened by the specified connection.

MQIACF_CONN_INFO_ALL
Connection information and information about those objects that the connection has open.

You cannot use *ConnInfoType* as a parameter to filter on.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ConnectionAttrs* except as noted and MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. You cannot use the value MQCNO_STANDARD_BINDING on the MQIACF_CONNECT_OPTIONS parameter with either the MQCFOP_CONTAINS or MQCFOP_EXCLUDES operator. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you filter on MQIACF_CONNECT_OPTIONS or MQIACF_OPEN_OPTIONS, in each case the filter value must have only 1 bit set.

If you specify an integer filter, you cannot also specify a byte string filter using the *ByteStringFilterCommand* parameter or a string filter using the *StringFilterCommand* parameter.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed

in *ConnectionAttrs*. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify a byte string filter using the *ByteStringFilterCommand* parameter or an integer filter using the *IntegerFilterCommand* parameter.

URDisposition **(MQCFIN)**
The unit of recovery disposition associated with the connection (parameter identifier: MQI_UR_DISP). This parameter is valid only on z/OS.

The value can be:

MQQSGD_ALL
Specifies that all connections must be returned.

MQQSGD_GROUP
Specifies that only connections with a GROUP unit of recovery disposition must be returned.

MQQSGD_Q_MGR
Specifies that only connections with a QMGR unit of recovery disposition must be returned.

Error code

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CONNECTION_ID_ERROR
Connection identifier not valid.

Inquire Connection (Response):

The response to the Inquire Connection (MQCMD_INQUIRE_CONNECTION) command consists of the response header followed by the *ConnectionId* structure and a set of attribute parameter structures determined by the value of *ConnInfoType* in the Inquire command.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

If the value of *ConnInfoType* was MQIACF_CONN_INFO_ALL, there is one message for each connection found with MQIACF_CONN_INFO_CONN, and *n* more messages per connection with MQIACF_CONN_INFO_HANDLE (where *n* is the number of objects that the connection has open).

Always returned:
ConnectionId, *ConnInfoType*

Always returned if *ConnInfoType* is MQIACF_CONN_INFO_HANDLE:
ObjectName, *ObjectType*, *QSGDisposition*

Returned if requested and *ConnInfoType* is MQIACF_CONN_INFO_CONN:
ApplDesc,*ApplTag*, *ApplType*, *ASID*, *AsynchronousState*, *ChannelName*, *ConnectionName*, *ConnectionOptions*, *OriginName*, *OriginUOWId*, *ProcessId*, *PSBName*, *PSTId*, *QMgrUOWId*, *StartUOWLogExtent*, *TaskNumber*, *ThreadId*, *TransactionId*, *UOWIdentifier*, *UOWLogStartDate*, *UOWLogStartTime*, *UOWStartDate*, *UOWStartTime*, *UOWState*, *UOWType*, *URDisposition*, *UserId*

Returned if requested and *ConnInfoType* is MQIACF_CONN_INFO_HANDLE:
AsynchronousState, *Destination*, *DestinationQueueManager*, *HandleState*, *OpenOptions*, *ReadAhead*, *SubscriptionID*,*SubscriptionName*, *TopicString*

Response data

ApplDesc **(MQCFST)**
Application description (parameter identifier: MQCACF_APPL_DESC).

The maximum length is MQ_APPL_DESC_LENGTH.

ApplTag **(MQCFST)**
Application tag (parameter identifier: MQCACF_APPL_TAG).

The maximum length is MQ_APPL_TAG_LENGTH.

ApplType **(MQCFIN)**
Application type (parameter identifier: MQIA_APPL_TYPE).

The value can be:

MQAT_QMGR
Queue manager process.

MQAT_CHANNEL_INITIATOR
Channel initiator.

MQAT_USER
User application.

MQAT_BATCH
Application using a batch connection (only on z/OS).

MQAT_RRS_BATCH
RRS-coordinated application using a batch connection (only on z/OS).

MQAT_CICS
CICS transaction (only on z/OS).

MQAT_IMS
IMS transaction (only on z/OS).

MQAT_SYSTEM_EXTENSION
Application performing an extension of function that is provided by the queue manager.

ASID **(MQCFST)**
Address space identifier (parameter identifier: MQCACF_ASID).

The four character address-space identifier of the application identified by *ApplTag*. It distinguishes duplicate values of *ApplTag*.

This parameter is valid only on z/OS.

The length of the string is MQ_ASID_LENGTH.

AsynchronousState **(MQCFIN)**
The state of asynchronous consumption on this handle (parameter identifier: MQIACF_ASYNC_STATE).

The value can be:

MQAS_NONE
If *ConnInfoType* is MQIACF_CONN_INFO_CONN, an MQCTL call has not been issued against the handle. Asynchronous message consumption cannot currently proceed on this connection. If *ConnInfoType* is MQIACF_CONN_INFO_HANDLE, an MQCB call has not been issued against this handle, so no asynchronous message consumption is configured on this handle.

MQAS_SUSPENDED
The asynchronous consumption callback has been suspended so that asynchronous message consumption cannot currently proceed on this handle. This situation can be either because an

MQCB or MQCTL call with *Operation* MQOP_SUSPEND has been issued against this object handle by the application, or because it has been suspended by the system. If it has been suspended by the system, as part of the process of suspending asynchronous message consumption the callback function is called with the reason code that describes the problem resulting in suspension. This reason code is reported in the *Reason* field in the MQCBC structure passed to the callback. In order for asynchronous message consumption to proceed, the application must issue an MQCB or MQCTL call with *Operation* MQOP_RESUME. This reason code can be returned if *ConnInfoType* is MQIACF_CONN_INFO_CONN or MQIACF_CONN_INFO_HANDLE.

MQAS_SUSPENDED_TEMPORARY
The asynchronous consumption callback has been temporarily suspended by the system so that asynchronous message consumption cannot currently proceed on this object handle. As part of the process of suspending asynchronous message consumption, the callback function is called with the reason code that describes the problem resulting in suspension. MQAS_SUSPENDED_TEMPORARY is reported in the *Reason* field in the MQCBC structure passed to the callback. The callback function is called again when asynchronous message consumption is resumed by the system when the temporary condition has been resolved. MQAS_SUSPENDED_TEMPORARY is returned only if ConnInfoType is MQIACF_CONN_INFO_HANDLE.

MQAS_STARTED
An MQCTL call with *Operation* MQOP_START has been issued against the connection handle so that asynchronous message consumption can proceed on this connection. MQAS_STARTED is returned only if *ConnInfoType* is MQIACF_CONN_INFO_CONN.

MQAS_START_WAIT
An MQCTL call with *Operation* MQOP_START_WAIT has been issued against the connection handle so that asynchronous message consumption can proceed on this connection. MQAS_START_WAIT is returned only if *ConnInfoType* is MQIACF_CONN_INFO_CONN.

MQAS_STOPPED
An MQCTL call with *Operation* MQOP_STOP has been issued against the connection handle so that asynchronous message consumption cannot currently proceed on this connection. MQAS_STOPPED is returned only if *ConnInfoType* is MQIACF_CONN_INFO_CONN.

MQAS_ACTIVE
An MQCB call has set up a function to call back to process messages asynchronously and the connection handle has been started so that asynchronous message consumption can proceed. MQAS_ACTIVE is returned only if *ConnInfoType* is MQIACF_CONN_INFO_HANDLE.

MQAS_INACTIVE
An MQCB call has set up a function to call back to process messages asynchronously but the connection handle has not yet been started, or has been stopped or suspended, so that asynchronous message consumption cannot currently proceed. MQAS_INACTIVE is returned only if ConnInfoType is MQIACF_CONN_INFO_HANDLE.

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ConnectionId **(MQCFBS)**
Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

The length of the string is MQ_CONNECTION_ID_LENGTH.

ConnectionName **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

ConnectionOptions **(MQCFIL)**
Connect options currently in force for the connection (parameter identifier: MQIACF_CONNECT_OPTIONS).

ConnInfoType **(MQCFIN)**
Type of information returned (parameter identifier: MQIACF_CONN_INFO_TYPE).

The value can be:

MQIACF_CONN_INFO_CONN
Generic information for the specified connection.

MQIACF_CONN_INFO_HANDLE
Information pertinent only to those objects opened by the specified connection.

Destination **(MQCFST)**
The destination queue for messages published to this subscription (parameter identifier MQCACF_DESTINATION).

This parameter is relevant only for handles of subscriptions to topics.

DestinationQueueManager **(MQCFST)**
The destination queue manager for messages published to this subscription (parameter identifier MQCACF_DESTINATION_Q_MGR).

This parameter is relevant only for handles of subscriptions to topics. If *Destination* is a queue hosted on the local queue manager, this parameter contains the local queue manager name. If *Destination* is a queue hosted on a remote queue manager, this parameter contains the name of the remote queue manager.

HandleState **(MQCFIN)**
State of the handle (parameter identifier: MQIACF_HANDLE_STATE).

The value can be:

MQHSTATE_ACTIVE
An API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when an MQGET WAIT call is in progress.

If there is an MQGET SIGNAL outstanding, then this situation does not mean, by itself, that the handle is active.

MQHSTATE_INACTIVE
No API call from this connection is currently in progress for this object. If the object is a queue, this condition can arise when no MQGET WAIT call is in progress.

ObjectName **(MQCFST)**
Object name (parameter identifier: MQCACF_OBJECT_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ObjectType **(MQCFIN)**
Object type (parameter identifier: MQIACF_OBJECT_TYPE).

If this parameter is a handle of a subscription to a topic, the SUBID parameter identifies the subscription and can be used with the Inquire Subscription command to find all the details about the subscription.

The value can be:

MQOT_Q
Queue.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q_MGR
Queue manager.

MQOT_CHANNEL
Channel.

MQOT_AUTH_INFO
Authentication information object.

MQOT_TOPIC
Topic.

OpenOptions **(MQCFIN)**
Open options currently in force for the object for connection (parameter identifier: MQIACF_OPEN_OPTIONS).

This parameter is not relevant for a subscription. Use the SUBID field of the DISPLAY SUB command to find all the details about the subscription.

OriginName **(MQCFST)**
Origin name (parameter identifier: MQCACF_ORIGIN_NAME).

Identifies the originator of the unit of recovery, except where *ApplType* is MQAT_RRS_BATCH when it is omitted.

This parameter is valid only on z/OS.

The length of the string is MQ_ORIGIN_NAME_LENGTH.

OriginUOWId **(MQCFBS)**
Origin UOW identifier (parameter identifier: MQBACF_ORIGIN_UOW_ID).

The unit of recovery identifier assigned by the originator. It is an 8-byte value.

This parameter is valid only on z/OS.

The length of the string is MQ_UOW_ID_LENGTH.

ProcessId **(MQCFIN)**
Process identifier (parameter identifier: MQIACF_PROCESS_ID).

PSBName **(MQCFST)**
Program specification block name (parameter identifier: MQCACF_PSB_NAME).

The 8-character name of the program specification block (PSB) associated with the running IMS transaction.

This parameter is valid only on z/OS.

The length of the string is MQ_PSB_NAME_LENGTH.

PSTId **(MQCFST)**
Program specification table identifier (parameter identifier: MQCACF_PST_ID).

The 4-character IMS program specification table (PST) region identifier for the connected IMS region.

This parameter is valid only on z/OS.

The length of the string is MQ_PST_ID_LENGTH.

QMgrUOWId **(MQCFBS)**
Unit of recovery identifier assigned by the queue manager (parameter identifier: MQBACF_Q_MGR_UOW_ID).

On z/OS platforms, this parameter is returned as a 6-byte RBA. On platforms other than z/OS, this parameter is an 8-byte transaction identifier.

The maximum length of the string is MQ_UOW_ID_LENGTH.

QSGDispositon **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid only on z/OS. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

ReadAhead **(MQCFIN)**
The read ahead connection status (parameter identifier: MQIA_READ_AHEAD).

The value can be:

MQREADA_NO
Read ahead for browsing messages, or of non-persistent messages is not enabled for the object that the connection has open.

MQREADA_YES
Read ahead for browsing messages, or of non-persistent messages is enabled for the object that the connection has open and is being used efficiently.

MQREADA_BACKLOG
Read ahead for browsing messages, or of non-persistent messages is enabled for this object. Read ahead is not being used efficiently because the client has been sent many messages which are not being consumed.

MQREADA_INHIBITED
Read ahead was requested by the application but has been inhibited because of incompatible options specified on the first MQGET call.

StartUOWLogExtent **(MQCFST)**
Name of the first extent needed to recover the transaction (parameter identifier: MQCACF_UOW_LOG_EXTENT_NAME).

The 8-character name of the program specification block (PSB) associated with the running IMS transaction.

This parameter is not valid on z/OS.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

SubscriptionID **(MQCFBS)**
The internal, all time unique identifier of the subscription (parameter identifier MQBACF_SUB_ID).

This parameter is relevant only for handles of subscriptions to topics.

Not all subscriptions can be seen using Inquire Connection; only those subscriptions that have current handles open to the subscriptions can be seen. Use the Inquire Subscription command to see all subscriptions.

SubscriptionName **(MQCFST)**
The unique subscription name of the application associated with the handle (parameter identifier MQCACF_SUB_NAME).

This parameter is relevant only for handles of subscriptions to topics. Not all subscriptions have a subscription name.

ThreadId **(MQCFIN)**
Thread identifier (parameter identifier: MQIACF_THREAD_ID).

TopicString **(MQCFST)**
Resolved topic string (parameter identifier: MQCA_TOPIC_STRING).

This parameter is relevant for handles with an ObjectType of MQOT_TOPIC. For any other object type, this parameter is blank.

TransactionId **(MQCFST)**
Transaction identifier (parameter identifier: MQCACF_TRANSACTION_ID).

The 4-character CICS transaction identifier.

This parameter is valid only on z/OS.

The maximum length of the string is MQ_TRANSACTION_ID_LENGTH.

UOWIdentifier **(MQCFBS)**
External unit of recovery identifier associated with the connection (parameter identifier: MQBACF_EXTERNAL UOW_ID).

This parameter is the recovery identifier for the unit of recovery. The value of *UOWType* determines its format.

The maximum length of the byte string is MQ_UOW_ID_LENGTH.

UOWLogStartDate **(MQCFST)**
Logged unit of work start date, in the form `yyyy-mm-dd` (parameter identifier: MQCACF_UOW_LOG_START_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

UOWLogStartTime **(MQCFST)**
Logged unit of work start time, in the form `hh.mm.ss` (parameter identifier: MQCACF_UOW_LOG_START_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

UOWStartDate **(MQCFST)**
Unit of work creation date (parameter identifier: MQCACF_UOW_START_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

UOWStartTime **(MQCFST)**
Unit of work creation time (parameter identifier: MQCACF_UOW_START_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

UOWState **(MQCFIN)**
State of the unit of work (parameter identifier: MQIACF_UOW_STATE).

The value can be:

MQUOWST_NONE
There is no unit of work.

MQUOWST_ACTIVE
The unit of work is active.

MQUOWST_PREPARED
The unit of work is in the process of being committed.

MQUOWST_UNRESOLVED
The unit of work is in the second phase of a two-phase commit operation. WebSphere MQ

holds resources on behalf of the unit of work and external intervention is required to resolve it. It might be as simple as starting the recovery coordinator (such as CICS, IMS, or RRS) or it might involve a more complex operation such as using the RESOLVE INDOUBT command. This value can occur only on z/OS.

UOWType **(MQCFIN)**
Type of external unit of recovery identifier as perceived by the queue manager (parameter identifier: MQIACF_UOW_TYPE).

The value can be:

MQUOWT_Q_MGR

MQUOWT_CICS

MQUOWT_RRS

MQUOWT_IMS

MQUOWT_XA

URDisposition **(MQCFIN)**
The unit of recovery disposition associated with the connection.

This parameter is valid only on z/OS.

The value can be:

MQQSGD_GROUP
This connection has a GROUP unit of recovery disposition.

MQQSGD_Q_MGR
This connection has a QMGR unit of recovery disposition.

UserId **(MQCFST)**
User identifier (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_MAX_USER_ID_LENGTH.

Inquire Entity Authority:

The Inquire Entity Authority (MQCMD_INQUIRE_ENTITY_AUTH) command inquires about authorizations of an entity to a specified object.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

The required parameters must all be passed in the following order: *Options*, *ObjectType*, *EntityType*, *EntityName*.

Options **(MQCFIN)**
Options to control the set of authority records that is returned (parameter identifier: MQIACF_AUTH_OPTIONS).

This parameter is required and you must set it to the value MQAUTHOPT_CUMULATIVE. It returns a set of authorities representing the cumulative authority that an entity has to a specified object.

If a user ID is a member of more than one group, this command displays the combined authorizations of all groups.

ObjectType **(MQCFIN)**
The type of object referred to by the profile (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

EntityType **(MQCFIN)**
Entity type (parameter identifier: MQIACF_ENTITY_TYPE).

The value can be:

MQZAET_GROUP
The value of the *EntityName* parameter refers to a group name.

MQZAET_PRINCIPAL
The value of the *EntityName* parameter refers to a principal name.

EntityName **(MQCFST)**
Entity name (parameter identifier: MQCACF_ENTITY_NAME).

Depending on the value of *EntityType*, this parameter is either:

- A principal name. This name is the name of a user for whom to retrieve authorizations to the specified object. On WebSphere MQ for Windows, the name of the principal can optionally include a domain name, specified in this format: `user@domain`.
- A group name. This name is the name of the user group on which to make the inquiry. You can specify one name only and this name must be the name of an existing user group.

  For WebSphere MQ for Windows only, the group name can optionally include a domain name, specified in the following formats:

  ```
  GroupName@domain
  domain\GroupName
  ```

The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

Optional parameters

ObjectName **(MQCFST)**
Object name (parameter identifier: MQCACF_OBJECT_NAME).

The name of the queue manager, queue, process definition, or generic profile on which to make the inquiry.

You must include a parameter if the *ObjectType* is not MQOT_Q_MGR. If you do not include this parameter, it is assumed that you are making an inquiry on the queue manager.

You cannot specify a generic object name although you can specify the name of a generic profile.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ProfileAttrs **(MQCFIL)**
Profile attributes (parameter identifier: MQIACF_AUTH_PROFILE_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCACF_ENTITY_NAME
Entity name.

MQIACF_AUTHORIZATION_LIST
Authorization list.

MQIACF_ENTITY_TYPE
Entity type.

MQIACF_OBJECT_TYPE
Object type.

ServiceComponent **(MQCFST)**
Service component (parameter identifier: MQCACF_SERVICE_COMPONENT).

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply.

If you omit this parameter, the authorization inquiry is made to the first installable component for the service.

The maximum length of the string is MQ_SERVICE_COMPONENT_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRC_UNKNOWN_ENTITY
User ID not authorized, or unknown.

MQRCCF_OBJECT_TYPE_MISSING
Object type missing.

Inquire Entity Authority (Response):

Each response to the Inquire Entity Authority (MQCMD_INQUIRE_AUTH_RECS) command consists of the response header followed by the *QMgrName*, *Options*, and *ObjectName* structures and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned:
ObjectName, *Options*, *QMgrName*

Returned if requested:
AuthorizationList, *EntityName*, *EntityType*, *ObjectType*

Response data

AuthorizationList **(MQCFIL)**
Authorization list(parameter identifier: MQIACF_AUTHORIZATION_LIST).

This list can contain zero or more authorization values. Each returned authorization value means that any user ID in the specified group or principal has the authority to perform the operation defined by that value. The value can be:

MQAUTH_NONE
The entity has authority set to 'none'.

MQAUTH_ALT_USER_AUTHORITY
Specify an alternate user ID on an MQI call.

MQAUTH_BROWSE
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

MQAUTH_CHANGE
Change the attributes of the specified object, using the appropriate command set.

MQAUTH_CLEAR
Clear a queue.

MQAUTH_CONNECT
Connect the application to the specified queue manager by issuing an MQCONN call.

MQAUTH_CREATE
Create objects of the specified type using the appropriate command set.

MQAUTH_DELETE
Delete the specified object using the appropriate command set.

MQAUTH_DISPLAY
Display the attributes of the specified object using the appropriate command set.

MQAUTH_INPUT
Retrieve a message from a queue by issuing an MQGET call.

MQAUTH_INQUIRE
Make an inquiry on a specific queue by issuing an MQINQ call.

MQAUTH_OUTPUT
Put a message on a specific queue by issuing an MQPUT call.

MQAUTH_PASS_ALL_CONTEXT
Pass all context.

MQAUTH_PASS_IDENTITY_CONTEXT
Pass the identity context.

MQAUTH_SET
Set attributes on a queue from the MQI by issuing an MQSET call.

MQAUTH_SET_ALL_CONTEXT
Set all context on a queue.

MQAUTH_SET_IDENTITY_CONTEXT
Set the identity context on a queue.

MQAUTH_CONTROL
For listeners and services, start and stop the specified channel, listener, or service.

For channels, start, stop, and ping the specified channel.

For topics, define, alter, or delete subscriptions.

MQAUTH_CONTROL_EXTENDED
Reset or resolve the specified channel.

MQAUTH_PUBLISH
Publish to the specified topic.

MQAUTH_SUBSCRIBE
Subscribe to the specified topic.

MQAUTH_RESUME
Resume a subscription to the specified topic.

MQAUTH_SYSTEM
Use queue manager for internal system operations.

MQAUTH_ALL
Use all operations applicable to the object.

MQAUTH_ALL_ADMIN
Use all administration operations applicable to the object.

MQAUTH_ALL_MQI
Use all MQI calls applicable to the object.

Use the *Count* field in the MQCFIL structure to determine how many values are returned.

EntityName **(MQCFST)**
Entity name (parameter identifier: MQCACF_ENTITY_NAME).

This parameter can either be a principal name or a group name.

The maximum length of the string is MQ_ENTITY_NAME_LENGTH.

EntityType **(MQCFIN)**
Entity type (parameter identifier: MQIACF_ENTITY_TYPE).

The value can be:

MQZAET_GROUP
The value of the *EntityName* parameter refers to a group name.

MQZAET_PRINCIPAL
The value of the *EntityName* parameter refers to a principal name.

MQZAET_UNKNOWN
On Windows, an authority record still exists from a previous queue manager which did not originally contain entity type information.

ObjectName **(MQCFST)**
Object name (parameter identifier: MQCACF_OBJECT_NAME).

The name of the queue manager, queue, process definition, or generic profile on which the inquiry is made.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

ObjectType **(MQCFIN)**
Object type (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

QMgrName **(MQCFST)**
Name of the queue manager on which the Inquire command is issued (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Inquire Namelist:

The Inquire Namelist (MQCMD_INQUIRE_NAMELIST) command inquires about the attributes of existing WebSphere MQ namelists.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters:
NamelistName

Optional parameters:
CommandScope, *IntegerFilterCommand*, *NamelistAttrs*, *QSGDisposition*, *StringFilterCommand*

Required parameters

NamelistName **(MQCFST)**
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

This parameter is the name of the namelist with attributes that are required. Generic namelist names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all namelists having names that start with the selected character string. An asterisk on its own matches all possible names.

The namelist name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *NamelistAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter for *NamelistType* (MQIA_NAMELIST_TYPE), you cannot also specify the *NamelistType* parameter.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

NamelistAttrs **(MQCFIL)**
Namelist attributes (parameter identifier: MQIACF_NAMELIST_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_NAMELIST_NAME
Name of namelist object.

MQCA_NAMELIST_DESC
Namelist description.

MQCA_NAMES
Names in the namelist.

MQCA_ALTERATION_DATE
The date on which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQIA_NAME_COUNT
Number of names in the namelist.

MQIA_NAMELIST_TYPE
Namelist type (valid only on z/OS)

NamelistType **(MQCFIN)**
Namelist attributes (parameter identifier: MQIA_NAMELIST_TYPE). This parameter applies to z/OS only.

Specifies the type of names in the namelist. The value can be:

MQNT_NONE
The names are of no particular type.

MQNT_Q
A namelist that holds a list of queue names.

MQNT_CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

MQNT_AUTH_INFO
The namelist is associated with SSL, and contains a list of authentication information object names.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *NamelistAttrs* except MQCA_NAMELIST_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Inquire Namelist (Response):

The response to the Inquire Namelist (MQCMD_INQUIRE_NAMELIST) command consists of the response header followed by the *NamelistName* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

If a generic namelist name was specified, one such message is generated for each namelist found.

Always returned:
NamelistName, *QSGDisposition*

Returned if requested:
AlterationDate, *AlterationTime*, *NameCount*, *NamelistDesc*, *NamelistType*, *Names*

Response data

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss`.

NameCount **(MQCFIN)**
Number of names in the namelist (parameter identifier: MQIA_NAME_COUNT).

The number of names contained in the namelist.

NamelistDesc **(MQCFST)**
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

NamelistName **(MQCFST)**
The name of the namelist definition (parameter identifier: MQCA_NAMELIST_NAME).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

NamelistType **(MQCFIN)**
Type of names in the namelist (parameter identifier: MQIA_NAMELIST_TYPE). This parameter applies to z/OS only.

Specifies the type of names in the namelist. The value can be:

MQNT_NONE
The names are of no particular type.

MQNT_Q
A namelist that holds a list of queue names.

MQNT_CLUSTER
A namelist that is associated with clustering, containing a list of the cluster names.

MQNT_AUTH_INFO
The namelist is associated with SSL, and contains a list of authentication information object names.

Names **(MQCFSL)**
A list of the names contained in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the *Count* field in the MQCFSL structure. The length of each name is given by the *StringLength* field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter applies only to z/OS. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Namelist Names:

The Inquire Namelist Names (MQCMD_INQUIRE_NAMELIST_NAMES) command inquires for a list of namelist names that match the generic namelist name specified.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

NamelistName **(MQCFST)**
Name of namelist (parameter identifier: MQCA_NAMELIST_NAME).

Generic namelist names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.

- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Namelist Names (Response):

The response to the Inquire Namelist Names (MQCMD_INQUIRE_NAMELIST_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified namelist name.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Additionally, on z/OS only, the *QSGDispositions* structure (with the same number of entries as the *NamelistNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *NamelistNames* structure.

Always returned:
NamelistNames, *QSGDispositions*

Returned if requested:
None

Response data

NamelistNames **(MQCFSL)**
List of namelist names (parameter identifier: MQCACF_NAMELIST_NAMES).

QSGDispositions **(MQCFIL)**
List of QSG dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter is valid only on z/OS. Possible values for fields in this structure are:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Process:

The Inquire Process (MQCMD_INQUIRE_PROCESS) command inquires about the attributes of existing WebSphere MQ processes.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

ProcessName **(MQCFST)**
Process name (parameter identifier: MQCA_PROCESS_NAME).

Generic process names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all processes having names that start with the selected character string. An asterisk on its own matches all possible names.

The process name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ProcessAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

ProcessAttrs **(MQCFIL)**
Process attributes (parameter identifier: MQIACF_PROCESS_ATTRS).

The attribute list might specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
The date at which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQCA_APPL_ID
Application identifier.

MQCA_ENV_DATA
Environment data.

MQCA_PROCESS_DESC
Description of process definition.

MQCA_PROCESS_NAME
Name of process definition.

MQCA_USER_DATA
User data.

MQIA_APPL_TYPE
Application type.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ProcessAttrs* except MQCA_PROCESS_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Inquire Process (Response):

The response to the Inquire Process (MQCMD_INQUIRE_PROCESS) command consists of the response header followed by the *ProcessName* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux systems	Windows
X	X	X

If a generic process name was specified, one such message is generated for each process found.

Always returned:
ProcessName, *QSGDisposition*

Returned if requested:
AlterationDate, *AlterationTime*, *ApplId*, *ApplType*, *EnvData*, *ProcessDesc*, *UserData*

Response data

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss`.

ApplId **(MQCFST)**
Application identifier (parameter identifier: MQCA_APPL_ID).

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

ApplType **(MQCFIN)**
Application type (parameter identifier: MQIA_APPL_TYPE).

The value can be:

MQAT_AIX
AIX application (same value as MQAT_UNIX)

MQAT_CICS
CICS transaction

MQAT_DOS
DOS client application

MQAT_MVS
z/OS application

MQAT_OS400
IBM i application

MQAT_QMGR
Queue manager

MQAT_UNIX
UNIX application

MQAT_WINDOWS
16-bit Windows application

MQAT_WINDOWS_NT
32-bit Windows application

integer System-defined application type in the range zero through 65 535 or a user-defined application type in the range 65 536 through 999 999 999

EnvData **(MQCFST)**
Environment data (parameter identifier: MQCA_ENV_DATA).

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc **(MQCFST)**
Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

ProcessName **(MQCFST)**
The name of the process definition (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

UserData **(MQCFST)**
User data (parameter identifier: MQCA_USER_DATA).

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Inquire Process Names:

The Inquire Process Names (MQCMD_INQUIRE_PROCESS_NAMES) command inquires for a list of process names that match the generic process name specified.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

ProcessName **(MQCFST)**
Name of process-definition for queue (parameter identifier: MQCA_PROCESS_NAME).

Generic process names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Process Names (Response):

The response to the Inquire Process Names (MQCMD_INQUIRE_PROCESS_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified process name.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Additionally, on z/OS only, a parameter structure, *QSGDispositions* (with the same number of entries as the *ProcessNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *ProcessNames* structure.

This response is not supported on Windows.

Always returned:
ProcessNames, *QSGDispositions*

Returned if requested:
None

Response data

ProcessNames **(MQCFSL)**
List of process names (parameter identifier: MQCACF_PROCESS_NAMES).

QSGDispositions **(MQCFIL)**
List of QSG dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter applies only to z/OS. Possible values for fields in this structure are:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Pub/Sub Status:

The Inquire Pub/Sub Status (MQCMD_INQUIRE_PUBSUB_STATUS) command inquires about the status of publish/subscribe connections.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

blank (or omit the parameter altogether)
The command is executed on the queue manager on which it was entered.

a queue manager name
The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

an asterisk (*)
The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use CommandScope as a parameter to filter on.

PubSubStatusAttrs **(MQCFIL)**
Publish/subscribe status attributes (parameter identifier: MQIACF_PUBSUB_STATUS_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQIACF_PUBSUB_STATUS
Hierarchy status.

MQIACF_PS_STATUS_TYPE
Hierarchy type.

Type **(MQCFIN)**
Type (parameter identifier: MQIACF_PS_STATUS_TYPE).

The type can specify one of the following:

MQPSST_ALL
Return status of both parent and child connections. MQPSST_ALL is the default value if the parameter is not specified.

MQPSST_LOCAL
Return local status information.

MQPSST_PARENT
Return status of the parent connection.

MQPSST_CHILD
Return status of the child connections.

Inquire Pub/Sub Status (Response):

The response to the Inquire publish/subscribe Status (MQCMD_INQUIRE_PUBSUB_STATUS) command consists of the response header followed by the attribute structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

A group of parameters is returned containing the following attributes: *Type*, *QueueManagerName*, and *Status*.

Always returned:
QueueManagerName, *Status*, *Type*

Returned if requested:
None

Response data

QueueManagerName **(MQCFST)**
Either the name of the local queue manager when TYPE is LOCAL, or the name of the hierarchically connected queue manager (parameter identifier: MQCA_Q_MGR_NAME).

Type **(MQCFIN)**
Type of status that is being returned (parameter identifier: MQIACF_PS_ STATUS_TYPE).

The value can be:

MQPSST_CHILD
Publish/subscribe status for a child hierarchical connection.

MQPSST_LOCAL
Publish/subscribe status for the local queue manager.

MQPSST_PARENT
Publish/subscribe status for the parent hierarchical connection.

Status **(MQCFIN)**
The status of the publish/subscribe engine or the hierarchical connection (parameter identifier: MQIACF_PUBSUB_STATUS).

When TYPE is LOCAL the following values can be returned:

MQPS_STATUS_ACTIVE
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe using the application programming interface and the queues that are monitored by the queued publish/subscribe interface appropriately.

MQPS_STATUS_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe using the application programming interface. The queued publish/subscribe interface is not running. Therefore, any message that is put to the queues monitored by the queued publish/subscribe interface is not acted upon by WebSphere MQ.

MQPS_STATUS_ERROR
The publish/subscribe engine has failed. Check your error logs to determine the reason for the failure.

MQPS_STATUS_INACTIVE
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface is not acted upon by WebSphere MQ.

If inactive and you want to start the publish/subscribe engine, on the Change Queue Manager command set PubSubMode to **MQPSM_ENABLED**.

MQPS_STATUS_STARTING
The publish/subscribe engine is initializing and is not yet operational.

MQPS_STATUS_STOPPING
The publish/subscribe engine is stopping.

When TYPE is PARENT, the following values can be returned:

MQPS_STATUS_ACTIVE
The connection with the parent queue manager is active.

MQPS_STATUS_ERROR
This queue manager is unable to initialize a connection with the parent queue manager because of a configuration error.

A message is produced in the queue manager logs to indicate the specific error. If you receive error message AMQ5821 or on z/OS systems CSQT821E, possible causes include:

- Transmit queue is full
- Transmit queue put disabled

If you receive error message AMQ5814 or on z/OS systems CSQT814E, take the following actions:

- Check that the parent queue manager is correctly specified.
- Ensure that broker is able to resolve the queue manager name of the parent broker.

To resolve the queue manager name, at least one of the following resources must be configured:

- A transmission queue with the same name as the parent queue manager name.
- A queue manager alias definition with the same name as the parent queue manager name.
- A cluster with the parent queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the parent queue manager name.
- A default transmission queue.

After you have set up the configuration correctly, modify the parent queue manager name to blank. Then set with the parent queue manager name.

MQPS_STATUS_REFUSED
The connection has been refused by the parent queue manager.

This situation might be caused by the parent queue manager already having another child queue manager of the same name as this queue manager.

Alternatively, the parent queue manager has used the RESET QMGR TYPE(PUBSUB) CHILD command to remove this queue manager as one of its children.

MQPS_STATUS_STARTING
The queue manager is attempting to request that another queue manager is its parent.

If the parent status remains in starting status without progressing to active status, take the following actions:

- Check that the sender channel to parent queue manager is running
- Check that the receiver channel from parent queue manager is running

MQPS_STATUS_STOPPING
The queue manager is disconnecting from its parent.

If the parent status remains in stopping status, take the following actions:

- Check that the sender channel to parent queue manager is running
- Check that the receiver channel from parent queue manager is running

When TYPE is CHILD, the following values can be returned:

MQPS_STATUS_ACTIVE
The connection with the parent queue manager is active.

MQPS_STATUS_ERROR
This queue manager is unable to initialize a connection with the parent queue manager because of a configuration error.

A message is produced in the queue manager logs to indicate the specific error. If you receive error message AMQ5821 or on z/OS systems CSQT821E, possible causes include:

- Transmit queue is full
- Transmit queue put disabled

If you receive error message AMQ5814 or on z/OS systems CSQT814E, take the following actions:

- Check that the child queue manager is correctly specified.
- Ensure that broker is able to resolve the queue manager name of the child broker.

To resolve the queue manager name, at least one of the following resources must be configured:

- A transmission queue with the same name as the child queue manager name.
- A queue manager alias definition with the same name as the child queue manager name.
- A cluster with the child queue manager a member of the same cluster as this queue manager.
- A cluster queue manager alias definition with the same name as the child queue manager name.
- A default transmission queue.

After you have set up the configuration correctly, modify the child queue manager name to blank. Then set with the child queue manager name.

MQPS_STATUS_STARTING
The queue manager is attempting to request that another queue manager is its parent.

If the child status remains in starting status without progressing to active status, take the following actions:

- Check that the sender channel to child queue manager is running
- Check that the receiver channel from child queue manager is running

MQPS_STATUS_STOPPING
The queue manager is disconnecting from its parent.

If the child status remains in stopping status, take the following actions:

- Check that the sender channel to child queue manager is running
- Check that the receiver channel from child queue manager is running

Inquire Queue:

Use the Inquire Queue command MQCMD_INQUIRE_Q to query the attributes of IBM WebSphere MQ queues.

HP Integrity NonStop Server	UNIX and Linux	Windows
✓	✓	✓

Required parameters

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk *; for example ABC*. It selects all queues having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

CFStructure **(MQCFST)**
Storage class (parameter identifier: MQCA_CF_STRUC_NAME). Specifies the name of the storage class. This parameter is valid only on z/OS.

This parameter specifies that eligible queues are limited to those having the specified *CFStructure* value. If this parameter is not specified, then all queues are eligible.

Generic CF structure names are supported. A generic name is a character string followed by an asterisk *; for example ABC*. It selects all CF structures having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

ClusterInfo **(MQCFIN)**
Cluster information (parameter identifier: MQIACF_CLUSTER_INFO).

This parameter requests that cluster information about these queues and other queues in the repository that match the selection criteria is displayed. The cluster information is displayed in addition to information about attributes of queues defined on this queue manager.

In this case, there might be multiple queues with the same name displayed. The cluster information is shown with a queue type of MQQT_CLUSTER.

You can set this parameter to any integer value, the value used does not affect the response to the command.

The cluster information is obtained locally from the queue manager.

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

This parameter specifies that eligible queues are limited to those having the specified *ClusterName* value. If this parameter is not specified, then all queues are eligible.

Generic cluster names are supported. A generic name is a character string followed by an asterisk *; for example ABC*. It selects all clusters having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist **(MQCFST)**
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

This parameter specifies that eligible queues are limited to those having the specified *ClusterNameList* value. If this parameter is not specified, then all queues are eligible.

Generic cluster namelists are supported. A generic name is a character string followed by an asterisk *; for example ABC*. It selects all cluster namelists having names that start with the selected character string. An asterisk on its own matches all possible names.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk "*". The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

*IntegerFilterCommand***(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *QAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter for *Qtype* or *PageSetID*, you cannot also specify the *Qtype* or *PageSetID* parameter.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

PageSetID **(MQCFIN)**
Page set identifier (parameter identifier: MQIA_PAGESET_ID). This parameter applies to z/OS only.

This parameter specifies that eligible queues are limited to those having the specified *PageSetID* value. If this parameter is not specified, then all queues are eligible.

QAttrs **(MQCFIL)**
Queue attributes (parameter identifier: MQIACF_Q_ATTRS).

The attribute list might specify the following value on its own. If the parameter is not specified, this value is the default:

MQIACF_ALL
All attributes.

You can also specify a combination of the parameters in the following table:

Table 100. Inquire Queue command, queue attributes

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQCA_ALTERATION_DATE The date on which the information was last altered	✓	✓	✓	✓	✓
MQCA_ALTERATION_TIME The time at which the information was last altered	✓	✓	✓	✓	✓
MQCA_BACKOUT_REQ_Q_NAME Excessive backout requeue name	✓	✓			
MQCA_BASE_NAME Name of queue that alias resolves to			✓		
MQCA_CF_STRUC_NAME Coupling facility structure name. This attribute is valid on z/OS only	✓	✓			
MQCA_CLUS_CHL_NAME The generic name of the cluster-sender channels that use this queue as a transmission queue.	✓	✓			
MQCA_CLUSTER_DATE Date when the definition became available to the local queue manager					✓
MQCA_CLUSTER_NAME Cluster name	✓		✓	✓	✓
MQCA_CLUSTER_NAMELIST Cluster namelist	✓		✓	✓	
MQCA_CLUSTER_Q_MGR_NAME Queue manager name that hosts the queue					✓
MQCA_CLUSTER_TIME Time when the definition became available to the local queue manager					✓
MQCA_CREATION_DATE Queue creation date	✓	✓			
MQCA_CREATION_TIME Queue creation time	✓	✓			
MQCA_CUSTOM The custom attribute for new features	✓	✓	✓	✓	✓
MQCA_INITIATION_Q_NAME Initiation queue name	✓	✓			

Table 100. Inquire Queue command, queue attributes (continued)

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQCA_PROCESS_NAME Name of process definition	✓	✓			
MQCA_Q_DESC Queue description	✓	✓	✓	✓	✓
MQCA_Q_MGR_IDENTIFIER Internally generated queue manager name					✓
MQCA_Q_NAME Queue name	✓	✓	✓	✓	✓
MQCA_REMOTE_Q_MGR_NAME Name of remote queue manager				✓	
MQCA_REMOTE_Q_NAME Name of remote queue as known locally on the remote queue manager				✓	
MQCA_STORAGE_CLASS Storage class. MQCA_STORAGE_CLASS is valid on z/OS only	✓	✓			
MQCA_TPIPE_NAME The **TPIPE** name used for communication with OTMA using the WebSphere MQ IMS Bridge	✓				
MQCA_TRIGGER_DATA Trigger data	✓	✓			
MQCA_XMIT_Q_NAME Transmission queue name				✓	
MQIA_ACCOUNTING_Q Accounting data collection	✓	✓			
MQIA_BACKOUT_THRESHOLD Backout threshold	✓	✓			
MQIA_BASE_TYPE Type of object	✓	✓	✓	✓	✓
MQIA_CLUSTER_Q_TYPE Cluster queue type					✓
MQIA_CLWL_Q_PRIORITY Cluster workload queue priority	✓		✓	✓	✓
MQIA_CLWL_Q_RANK Cluster workload queue rank	✓		✓	✓	✓

Table 100. Inquire Queue command, queue attributes (continued)

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQIA_CLWL_USEQ Cluster workload use remote setting	✓				
MQIA_CURRENT_Q_DEPTH Number of messages on queue	✓				
MQIA_DEF_BIND Default binding	✓		✓	✓	✓
MQIA_DEF_INPUT_OPEN_OPTION Default open-for-input option	✓	✓			
MQIA_DEF_PERSISTENCE Default message persistence	✓	✓	✓	✓	✓
MQIA_DEF_PRIORITY Default message priority	✓	✓	✓	✓	✓
MQIA_DEF_PUT_RESPONSE_TYPE Default put response type	✓	✓	✓	✓	✓
MQIA_DEF_READ_AHEAD Default put response type	✓	✓	✓	✓	✓
MQIA_DEFINITION_TYPE Queue definition type	✓	✓			
MQIA_DIST_LISTS Distribution list support. MQIA_DIST_LISTS is not valid on z/OS	✓	✓			
MQIA_HARDEN_GET_BACKOUT Whether to harden backout count	✓	✓			
MQIA_INDEX_TYPE Index type. This attribute is valid on z/OS only.	✓	✓			
MQIA_INHIBIT_GET Whether get operations are allowed	✓	✓	✓		
MQIA_INHIBIT_PUT Whether put operations are allowed	✓	✓	✓	✓	✓
MQIA_MAX_MSG_LENGTH Maximum message length	✓	✓			
MQIA_MAX_Q_DEPTH Maximum number of messages allowed on queue	✓	✓			

Table 100. Inquire Queue command, queue attributes (continued)

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQIA_MONITORING_Q Online monitoring data collection	✓	✓			
MQIA_MSG_DELIVERY_SEQUENCE Whether message priority is relevant	✓	✓			
MQIA_NPM_CLASS Level of reliability assigned to non-persistent messages that are put to the queue	✓	✓			
MQIA_OPEN_INPUT_COUNT Number of MQOPEN calls that have the queue open for input	✓				
MQIA_OPEN_OUTPUT_COUNT Number of MQOPEN calls that have the queue open for output	✓				
MQIA_PAGESET_ID Page set identifier	✓				
MQIA_PROPERTY_CONTROL Property control attribute	✓	✓	✓		
MQIA_Q_DEPTH_HIGH_EVENT Control attribute for queue depth high events. You cannot use MQIA_Q_DEPTH_HIGH_EVENT as a filter attribute.	✓	✓			
MQIA_Q_DEPTH_HIGH_LIMIT High limit for queue depth	✓	✓			
MQIA_Q_DEPTH_LOW_EVENT Control attribute for queue depth low events. You cannot use MQIA_Q_DEPTH_LOW_EVENT as a filter attribute.	✓	✓			
MQIA_Q_DEPTH_LOW_LIMIT Low limit for queue depth	✓	✓			
MQIA_Q_DEPTH_MAX_EVENT Control attribute for queue depth max events	✓	✓			
MQIA_Q_SERVICE_INTERVAL Limit for queue service interval	✓	✓			

Table 100. Inquire Queue command, queue attributes (continued)

	Local queue	Model queue	Alias queue	Remote queue	Cluster queue
MQIA_Q_SERVICE_INTERVAL_ EVENT Control attribute for queue service interval events	✓	✓			
MQIA_Q_TYPE Queue type	✓	✓	✓	✓	✓
MQIA_RETENTION_INTERVAL Queue retention interval	✓	✓			
MQIA_SCOPE Queue definition scope. MQIA_SCOPE is not valid on z/OS or IBM i	✓		✓	✓	
MQIA_SHAREABILITY Whether queue can be shared	✓	✓			
MQIA_STATISTICS_Q Statistics data collection. MQIA_STATISTICS_Q is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.	✓	✓			
MQIA_TRIGGER_CONTROL Trigger control	✓	✓			
MQIA_TRIGGER_DEPTH Trigger depth	✓	✓			
MQIA_TRIGGER_MSG_PRIORITY Threshold message priority for triggers	✓	✓			
MQIA_TRIGGER_MTYPE Trigger type	✓	✓			
MQIA_USAGE Usage	✓	✓			

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned. The meaning of "the disposition of an object" is where the object is defined and how it behaves. The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. In a shared queue manager environment, if the command is run on the queue manager where it was issued, MQQSGD_LIVE also returns information for objects defined with MQQSGD_SHARED. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

In a shared queue manager environment, if the command is run on the queue manager where it was issued, MQQSGD_ALL also displays information for objects defined with MQQSGD_GROUP or MQQSGD_SHARED.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names, with different dispositions.

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined with either MQQSGD_Q_MGR or MQQSGD_COPY.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED. MQQSGD_SHARED is permitted only in a shared queue environment.

You cannot use *QSGDisposition* as a parameter to filter on.

QType **(MQCFIN)**
Queue type (parameter identifier: MQIA_Q_TYPE).

If this parameter is present, eligible queues are limited to the specified type. Any attribute selector specified in the *QAttrs* list which is valid only for queues of a different type or types is ignored; no error is raised.

If this parameter is not present, or if MQQT_ALL is specified, queues of all types are eligible. Each attribute specified must be a valid queue attribute selector. The attribute can apply to some of the queues returned. It does not have to apply to all the queues. Queue attribute selectors that are valid but not applicable to the queue are ignored, no error messages occur and no attribute is returned. The following lists contains the value of all valid queue attribute selectors:

MQQT_ALL
All queue types.

MQQT_LOCAL
Local queue.

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_CLUSTER
Cluster queue.

MQQT_MODEL
Model queue definition.

Note: On platforms other than z/OS, if this parameter is present, it must occur immediately after the *QName* parameter.

StorageClass **(MQCFST)**
Storage class (parameter identifier: MQCA_STORAGE_CLASS). Specifies the name of the storage class. This parameter is valid only on z/OS.

This parameter specifies that eligible queues are limited to those having the specified *StorageClass* value. If this parameter is not specified, then all queues are eligible.

Generic names are supported. A generic name is a character string followed by an asterisk *; for example ABC*. It selects all storage classes having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *QAttrs* except MQCA_Q_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter for *ClusterName*, *ClusterNameList*, *StorageClass*, or *CFStructure*, you cannot also specify that as a parameter.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

Inquire Queue (Response):

The response to the Inquire Queue command MQCMD_INQUIRE_Q consists of the response header followed by the *QName* structure. On z/OS only, response includes the *QSGDisposition* structure, and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
✓	✓	✓

If a generic queue name was specified, or cluster queues requested, by setting either MQQT_CLUSTER or MQIACF_CLUSTER_INFO, one message is generated for each queue found.

Always returned:
QName, *QSGDisposition*, *QType*

Returned if requested:
AlterationDate, *AlterationTime*, *BackoutRequeueName*, *BackoutThreshold*, *BaseQName*, *CFStructure*, *ClusterChannelName*, *ClusterDate*, *ClusterName*, *ClusterNamelist*, *ClusterQType*, *ClusterTime*, *CLWLQueuePriority*, *CLWLQueueRank*, *CLWLUseQ*, *CreationDate*, *CreationTime*, *CurrentQDepth*, *Custom*, *DefaultPutResponse*, *DefBind*, *DefinitionType*, *DefInputOpenOption*, *DefPersistence*, *DefPriority*, *DefReadAhead*, *DistLists*, *HardenGetBackout*, *IndexType*, *InhibitGet*, *InhibitPut*, *InitiationQName*, *MaxMsgLength*, *MaxQDepth*, *MsgDeliverySequence*, *NonPersistentMessageClass*, *OpenInputCount*, *OpenOutputCount*, *PageSetID*, *ProcessName*, *PropertyControl*, *QDepthHighEvent*, *QDepthHighLimit*, *QDepthLowEvent*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDesc*, *QMgrIdentifier*, *QMgrName*, *QServiceInterval*, *QServiceIntervalEvent*, *QueueAccounting*, *QueueMonitoring*, *QueueStatistics*, *RemoteQMgrName*, *RemoteQName*, *RetentionInterval*, *Scope*, *Shareability*, *StorageClass*, *TpipeNames*, *TriggerControl*, *TriggerData*, *TriggerDepth*, *TriggerMsgPriority*, *TriggerType*, *Usage*, *XmitQName*

Response data

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form yyyy-mm-dd.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form hh.mm.ss.

BackoutRequeueName **(MQCFST)**
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

BackoutThreshold **(MQCFIN)**
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

BaseQName **(MQCFST)**
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

The name of a queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CFStructure **(MQCFST)**
Coupling facility structure name (parameter identifier: MQCA_CF_STRUC_NAME). This parameter applies to z/OS only.

Specifies the name of the coupling facility structure where you want to store messages when you use shared queues.

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

ClusterChannelName **(MQCFST)**
Cluster-sender channel name (parameter identifier: MQCA_CLUS_CHL_NAME).

ClusterChannelName is the generic name of the cluster-sender channels that use this queue as a transmission queue.

The maximum length of the channel name is: MQ_CHANNEL_NAME_LENGTH.

ClusterDate **(MQCFST)**
Cluster date (parameter identifier: MQCA_CLUSTER_DATE).

The date on which the information became available to the local queue manager, in the form yyyy-mm-dd.

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterNamelist **(MQCFST)**
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

ClusterQType **(MQCFIN)**
Cluster queue type (parameter identifier: MQIA_CLUSTER_Q_TYPE).

The value can be:

MQCQT_LOCAL_Q
The cluster queue represents a local queue.

MQCQT_ALIAS_Q
The cluster queue represents an alias queue.

MQCQT_REMOTE_Q
The cluster queue represents a remote queue.

MQCQT_Q_MGR_ALIAS
The cluster queue represents a queue manager alias.

ClusterTime **(MQCFST)**
Cluster time (parameter identifier: MQCA_CLUSTER_TIME).

The time at which the information became available to the local queue manager, in the form hh.mm.ss.

CLWLQueuePriority **(MQCFIN)**
Cluster workload queue priority (parameter identifier: MQIA_CLWL_Q_PRIORITY).

Priority of the queue in cluster workload management. The value is in the range zero through 9, where zero is the lowest priority and 9 is the highest.

CLWLQueueRank **(MQCFIN)**
Cluster workload queue rank (parameter identifier: MQIA_CLWL_Q_RANK).

Rank of the queue in cluster workload management. The value is in the range zero through 9, where zero is the lowest rank and 9 is the highest.

CLWLUseQ **(MQCFIN)**
Cluster workload queue rank (parameter identifier: MQIA_CLWL_USEQ).

The value can be:

MQCLWL_USEQ_AS_Q_MGR
Use the value of the *CLWLUseQ* parameter on the queue manager's definition.

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CreationDate **(MQCFST)**
Queue creation date, in the form yyyy-mm-dd (parameter identifier: MQCA_CREATION_DATE).

The maximum length of the string is MQ_CREATION_DATE_LENGTH.

CreationTime **(MQCFST)**
Creation time, in the form hh.mm.ss (parameter identifier: MQCA_CREATION_TIME).

The maximum length of the string is MQ_CREATION_TIME_LENGTH.

CurrentQDepth **(MQCFIN)**
Current queue depth (parameter identifier: MQIA_CURRENT_Q_DEPTH).

Custom **(MQCFST)**
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes are named. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form NAME(VALUE).

This description is updated when features using this attribute are introduced.

DefaultPutResponse **(MQCFIN)**
Default put response type definition (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The parameter specifies the type of response to be used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF. The value can be:

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

DefBind **(MQCFIN)**
Default binding (parameter identifier: MQIA_DEF_BIND).

The value can be:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

DefinitionType **(MQCFIN)**
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

The value can be:

MQQDT_PREDEFINED
Predefined permanent queue.

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_SHARED_DYNAMIC
Dynamically defined shared queue. This option is available on z/OS only.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

DefInputOpenOption **(MQCFIN)**
Default input open option for defining whether queues can be shared (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

The value can be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

DefPersistence **(MQCFIN)**
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

The value can be:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority **(MQCFIN)**
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

DefReadAhead **(MQCFIN)**
Default read ahead (parameter identifier: MQIA_DEF_READ_AHEAD).

Specifies the default read ahead behavior for non-persistent messages delivered to the client.

The value can be:

MQREADA_NO
Non-persistent messages are not sent ahead to the client before an application requests them. A maximum of one non-persistent message can be lost if the client ends abnormally.

MQREADA_YES
Non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages it is sent.

MQREADA_DISABLED
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

DistLists **(MQCFIN)**
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

The value can be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

This parameter is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Windows, and Linux.

HardenGetBackout **(MQCFIN)**
Harden backout, or not: (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

The value can be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count may not be remembered.

IndexType **(MQCFIN)**
Index type (parameter identifier: MQIA_INDEX_TYPE). This parameter applies to z/OS only.

Specifies the type of index maintained by the queue manager to expedite MQGET operations on the queue. The value can be:

MQIT_NONE
No index.

MQIT_MSG_ID
The queue is indexed using message identifiers.

MQIT_CORREL_ID
The queue is indexed using correlation identifiers.

MQIT_MSG_TOKEN
The queue is indexed using message tokens.

MQIT_GROUP_ID
The queue is indexed using group identifiers.

InhibitGet **(MQCFIN)**
Get operations are allowed or inhibited: (parameter identifier: MQIA_INHIBIT_GET).

The value can be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut **(MQCFIN)**
Putt operations are allowed or inhibited: (parameter identifier: MQIA_INHIBIT_PUT).

The value can be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

InitiationQName **(MQCFST)**
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

MaxQDepth **(MQCFIN)**
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

MsgDeliverySequence **(MQCFIN)**
Messages ordered by priority or sequence: (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value can be:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

NonPersistentMessageClass **(MQCFIN)**
The level of reliability assigned to non-persistent messages that are put to the queue (parameter identifier: MQIA_NPM_CLASS).

Specifies the circumstances under which non-persistent messages put to the queue may be lost. The value can be:

MQNPM_CLASS_NORMAL
Non-persistent messages are limited to the lifetime of the queue manager session. They are discarded in the event of a queue manager restart. MQNPM_CLASS_NORMAL is the default value.

MQNPM_CLASS_HIGH
The queue manager attempts to retain non-persistent messages for the lifetime of the queue. Non-persistent messages may still be lost in the event of a failure.

OpenInputCount **(MQCFIN)**
Number of MQOPEN calls that have the queue open for input (parameter identifier: MQIA_OPEN_INPUT_COUNT).

OpenOutputCount **(MQCFIN)**
Number of MQOPEN calls that have the queue open for output (parameter identifier: MQIA_OPEN_OUTPUT_COUNT).

PageSetID **(MQCFIN)**
Page set identifier (parameter identifier: MQIA_PAGESET_ID).

Specifies the identifier of the page set on which the queue resides.

This parameter applies to z/OS only when the queue is actively associated with a page set.

ProcessName **(MQCFST)**
Name of process definition for queue (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

PropertyControl **(MQCFIN)**
Property control attribute (parameter identifier MQIA_PROPERTY_CONTROL).

Specifies how message properties are handled for messages that are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option. The value can be:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of **mcd.**, **jms.**, **usr.** or **mqext.**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except properties contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

MQPROP_COMPATIBILITY is the default value. It allows applications which expect JMS-related properties to be in an MQRFH2 header in the message data to continue to work unmodified.

MQPROP_NONE
All properties of the message are removed from the message before the message is sent to the remote queue manager. Properties in the message descriptor (or extension) are not removed.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties are placed in one or more MQRFH2 headers in the message data. Properties in the message descriptor (or extension) are not placed in MQRFH2 headers.

MQPROP_FORCE_MQRFH2
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle.

A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGETcall is ignored. Properties of the message are not accessible via the message handle.

This parameter is applicable to local, alias, and model queues.

QDepthHighEvent **(MQCFIN)**
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighLimit **(MQCFIN)**
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth High event.

QDepthLowEvent **(MQCFIN)**
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowLimit **(MQCFIN)**
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

QDepthMaxEvent **(MQCFIN)**
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDesc **(MQCFST)**
Queue description (parameter identifier: MQCA_Q_DESC).

The maximum length of the string is MQ_Q_DESC_LENGTH.

QMgrIdentifier **(MQCFST)**
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

QMgrName **(MQCFST)**
Name of local queue manager (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QServiceInterval **(MQCFIN)**
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events.

QServiceIntervalEvent **(MQCFIN)**
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

The value can be:

MQQSIE_HIGH
Queue Service Interval High events enabled.

MQQSIE_OK
Queue Service Interval OK events enabled.

MQQSIE_NONE
No queue service interval events enabled.

QSGDisposition **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). *QSGDisposition* is valid only on z/OS. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

QType **(MQCFIN)**
Queue type (parameter identifier: MQIA_Q_TYPE).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_CLUSTER
Cluster queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

QueueAccounting **(MQCFIN)**
Controls the collection of accounting (thread-level and queue-level accounting) data (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_Q_MGR
The collection of accounting data for the queue is performed based upon the setting of the *QueueAccounting* parameter on the queue manager.

MQMON_OFF
Do not collect accounting data for the queue.

MQMON_ON
Collect accounting data for the queue.

QueueMonitoring **(MQCFIN)**
Online monitoring data collection (parameter identifier: MQIA_MONITORING_Q).

The value can be:

MQMON_OFF
Online monitoring data collection is turned off for this queue.

MQMON_Q_MGR
The value of the queue manager's *QueueMonitoring* parameter is inherited by the queue.

MQMON_LOW
Online monitoring data collection is turned on, with a low rate of data collection, for this queue unless *QueueMonitoring* for the queue manager is MQMON_NONE.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate rate of data collection, for this queue unless *QueueMonitoring* for the queue manager is MQMON_NONE.

MQMON_HIGH
Online monitoring data collection is turned on, with a high rate of data collection, for this queue unless *QueueMonitoring* for the queue manager is MQMON_NONE.

QueueStatistics **(MQCFIN)**
Controls the collection of statistics data (parameter identifier: MQIA_STATISTICS_Q).

The value can be:

MQMON_Q_MGR
The collection of statistics data for the queue is performed based upon the setting of the *QueueStatistics* parameter on the queue manager.

MQMON_OFF
Do not collect statistics data for the queue.

MQMON_ON
Collect statistics data for the queue unless *QueueStatistics* for the queue manager is MQMON_NONE.

This parameter is valid only on IBM i, UNIX systems, and Windows.

RemoteQMgrName **(MQCFST)**
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteQName **(MQCFST)**
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

RetentionInterval **(MQCFIN)**
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

Scope **(MQCFIN)**
Scope of the queue definition (parameter identifier: MQIA_SCOPE).

The value can be:

MQSCO_Q_MGR
Queue-manager scope.

MQSCO_CELL
Cell scope.

This parameter is not valid on IBM i or z/OS.

Shareability **(MQCFIN)**
The queue can be shared, or not: (parameter identifier: MQIA_SHAREABILITY).

The value can be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

StorageClass **(MQCFST)**
Storage class (parameter identifier: MQCA_STORAGE_CLASS). This parameter applies to z/OS only.

Specifies the name of the storage class.

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

TpipeNames **(MQCFSL)**
TPIPE names (parameter identifier: MQCA_TPIPE_NAME). This parameter applies to local queues on z/OS only.

Specifies the TPIPE names used for communication with OTMA via the WebSphere MQ IMS bridge, if the bridge is active.

The maximum length of the string is MQ_TPIPE_NAME_LENGTH.

TriggerControl **(MQCFIN)**
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

The value can be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerData **(MQCFST)**
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

TriggerDepth **(MQCFIN)**
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

TriggerMsgPriority **(MQCFIN)**
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

TriggerType **(MQCFIN)**
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

The value can be:

MQTT_NONE
No trigger messages.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_EVERY
Trigger message for every message.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

Usage **(MQCFIN)**
Usage (parameter identifier: MQIA_USAGE).

The value can be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

XmitQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Inquire Queue Manager:

The Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command inquires about the attributes of a queue manager.

HP Integrity NonStop Server	UNIX and Linux	Windows
✓	✓	✓

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following values:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment. The command server must be enabled.
- An asterisk "*". The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

QMgrAttrs **(MQCFIL)**
Queue manager attributes (parameter identifier: MQIACF_Q_MGR_ATTRS).

The attribute list might specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

Or a combination of the following values:

MQCA_ALTERATION_DATE
Date at which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCA_CHANNEL_AUTO_DEF_EXIT
Automatic channel definition exit name. MQCA_CHANNEL_AUTO_DEF_EXIT is not valid on z/OS.

MQCA_CLUSTER_WORKLOAD_DATA
Data passed to the cluster workload exit.

MQCA_CLUSTER_WORKLOAD_EXIT
Name of the cluster workload exit.

MQCA_COMMAND_INPUT_Q_NAME
System command input queue name.

MQCA_CUSTOM
The custom attribute for new features.

MQCA_DEAD_LETTER_Q_NAME
Name of dead-letter queue.

MQCA_DEF_XMIT_Q_NAME
Default transmission queue name.

MQCA_DNS_GROUP
The name of the group that the TCP listener handling inbound transmissions for the queue-sharing group must join when using Workload Manager for Dynamic Domain Name Services support (DDNS). MQCA_DNS_GROUP is valid on z/OS only.

MQCA_IGQ_USER_ID
Intra-group queuing user identifier. This parameter is valid on z/OS only.

MQCA_LU_GROUP_NAME
Generic LU name for the LU 6.2 listener. MQCA_LU_GROUP_NAME is valid on z/OS only.

MQCA_LU_NAME
LU name to use for outbound LU 6.2 transmissions. MQCA_LU_NAME is valid on z/OS only.

MQCA_LU62_ARM_SUFFIX
APPCPM suffix. MQCA_LU62_ARM_SUFFIX is valid on z/OS only.

MQCA_PARENT
The name of the hierarchically connected queue manager that is nominated as the parent of this queue manager.

MQCA_Q_MGR_DESC
Queue manager description.

MQCA_Q_MGR_IDENTIFIER
Internally generated unique queue manager name.

MQCA_Q_MGR_NAME
Name of local queue manager.

MQCA_QSG_NAME
Queue sharing group name. This parameter attribute is valid onz/OS only.

MQCA_REPOSITORY_NAME
Cluster name for the queue manager repository.

MQCA_REPOSITORY_NAMELIST
Name of the list of clusters for which the queue manager is providing a repository manager service.

MQCA_SSL_CRL_NAMELIST
SSL certificate revocation location namelist.

MQCA_SSL_CRYPTO_HARDWARE
Parameters to configure the SSL cryptographic hardware. This parameter is supported on UNIX, Linux, and Windows platforms only.

MQCA_SSL_KEY_REPOSITORY
Location and name of the SSL key repository.

MQCA_TCP_NAME
Name of the TCP/IP system that you are using. MQCA_TCP_NAME is valid on z/OS only.

MQCA_VERSION
The version of the IBM WebSphere MQ installation, the queue manager is associated with. The version has the format VVRRMMFF:

VV: Version

RR: Release

MM: Maintenance level

FF: Fix level

MQIA_ACCOUNTING_CONN_OVERRIDE
Specifies whether the settings of the *MQIAccounting* and *QueueAccounting* queue manager parameters can be overridden. MQIA_ACCOUNTING_CONN_OVERRIDE is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIA_ACCOUNTING_INTERVAL
Intermediate accounting data collection interval. MQIA_ACCOUNTING_INTERVAL is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIA_ACCOUNTING_MQI
Specifies whether accounting information is to be collected for MQI data.
MQIA_ACCOUNTING_MQI is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIA_ACCOUNTING_Q
Accounting data collection for queues.

MQIA_ACTIVE_CHANNELS
Maximum number of channels that can be active at any time. MQIA_ACTIVE_CHANNELS is valid on z/OS only.

MQIA_ACTIVITY_CONN_OVERRIDE
Specifies whether the value of application activity trace can be overridden.

MQIA_ACTIVITY_RECORDING
Specifies whether activity reports can be generated.

MQIA_ACTIVITY_TRACE
Specifies whether application activity trace reports can be generated.

MQIA_ADOPTNEWMCA_CHECK
Elements checked to determine whether an MCA must be adopted when a new inbound channel is detected with the same name as an MCA that is already active.
MQIA_ADOPTNEWMCA_CHECK is valid on z/OS only.

MQIA_ADOPTNEWMCA_TYPE
Specifies whether an orphaned instance of an MCA must be restarted automatically when a new inbound channel request matching the *AdoptNewMCACheck* parameter is detected.
MQIA_ADOPTNEWMCA_TYPE is valid on z/OS only.

MQIA_AUTHORITY_EVENT
Control attribute for authority events.

MQIA_BRIDGE_EVENT
Control attribute for IMS Bridge events. MQIA_BRIDGE_EVENT is valid only on z/OS.

MQIA_CERT_VAL_POLICY
Specifies which SSL/TLS certificate validation policy is used to validate digital certificates received from remote partner systems. This attribute controls how strictly the certificate chain validation conforms to industry security standards. MQIA_CERT_VAL_POLICY is valid on only UNIX, Linux, and Windows. For more information, see Certificate validation policies in WebSphere MQ.

MQIA_CHANNEL_AUTO_DEF
Control attribute for automatic channel definition. MQIA_CHANNEL_AUTO_DEF is not valid on z/OS.

MQIA_CHANNEL_AUTO_DEF_EVENT
Control attribute for automatic channel definition events. MQIA_CHANNEL_AUTO_DEF_EVENT is not valid on z/OS.

MQIA_CHANNEL_EVENT
Control attribute for channel events.

MQIA_CHINIT_ADAPTERS
Number of adapter subtasks to use for processing IBM WebSphere MQ calls. MQIA_CHINIT_ADAPTERS is valid on z/OS only.

MQIA_CHINIT_CONTROL
Start channel initiator automatically when queue manager starts.

MQIA_CHINIT_DISPATCHERS
Number of dispatchers to use for the channel initiator. MQIA_CHINIT_DISPATCHERS is valid on z/OS only.

MQIA_CHINIT_SERVICE_PARM
Reserved for use by IBM. MQIA_CHINIT_SERVICE_PARM is valid only on z/OS.

MQIA_CHINIT_TRACE_AUTO_START
Specifies whether the channel initiator trace must start automatically. MQIA_CHINIT_TRACE_AUTO_START is valid on z/OS only.

MQIA_CHINIT_TRACE_TABLE_SIZE
Size, in megabytes, of the trace data space of the channel initiator. MQIA_CHINIT_TRACE_TABLE_SIZE is valid on z/OS only.

MQIA_CHLAUTH_RECORDS
Control attribute for checking of channel authentication records.

MQIA_CLUSTER_WORKLOAD_LENGTH
Maximum length of the message passed to the cluster workload exit.

MQIA_CLWL_MRU_CHANNELS
Cluster workload most recently used channels.

MQIA_CLWL_USEQ
Cluster workload remote queue use.

MQIA_CMD_SERVER_CONTROL
Start command server automatically when queue manager starts.

MQIA_CODED_CHAR_SET_ID
Coded character set identifier.

MQIA_COMMAND_EVENT
Control attribute for command events.

MQIA_COMMAND_LEVEL
Command level supported by queue manager.

MQIA_CONFIGURATION_EVENT
Control attribute for configuration events.

MQIA_CPI_LEVEL
Reserved for use by IBM.

MQIA_DEF_CLUSTER_XMIT_Q_TYPE
Default transmission queue type to be used for cluster-sender channels. This parameter is not valid on z/OS.

MQIA_DIST_LISTS
Distribution list support. This parameter is not valid on z/OS.

MQIA_DNS_WLM
Specifies whether the TCP listener that handles inbound transmissions for the queue-sharing group must register with Workload Manager (WLM) for DDNS. MQIA_DNS_WLM is valid on z/OS only.

MQIA_EXPIRY_INTERVAL
Expiry interval. This parameter is valid on z/OS only.

MQIA_GROUP_UR
Control attribute for whether transactional applications can connect with a GROUP unit of recovery disposition. This parameter is valid only on z/OS.

MQIA_IGQ_PUT_AUTHORITY
Intra-group queuing put authority. This parameter is valid on z/OS only.

MQIA_INHIBIT_EVENT
Control attribute for inhibit events.

MQIA_INTRA_GROUP_QUEUING
Intra-group queuing support. This parameter is valid on z/OS only.

MQIA_IP_ADDRESS_VERSION
IP address version selector.

MQIA_LISTENER_TIMER
Listener restart interval. MQIA_LISTENER_TIMER is valid on z/OS only.

MQIA_LOCAL_EVENT
Control attribute for local events.

MQIA_LOGGER_EVENT
Control attribute for recovery log events.

MQIA_LU62_CHANNELS
Maximum number of LU 6.2 channels. MQIA_LU62_CHANNELS is valid on z/OS only.

MQIA_MSG_MARK_BROWSE_INTERVAL
Interval for which messages that were browsed, remain marked.

MQIA_MAX_CHANNELS
Maximum number of channels that can be current. MQIA_MAX_CHANNELS is valid on z/OS only.

MQIA_MAX_HANDLES
Maximum number of handles.

MQIA_MAX_MSG_LENGTH
Maximum message length.

MQIA_MAX_PRIORITY
Maximum priority.

MQIA_MAX_PROPERTIES_LENGTH
Maximum properties length.

MQIA_MAX_UNCOMMITTED_MSGS
Maximum number of uncommitted messages within a unit of work.

MQIA_MONITORING_AUTO_CLUSSDR
Default value of the *ChannelMonitoring* attribute of automatically defined cluster-sender channels.

MQIA_MONITORING_CHANNEL
Specifies whether channel monitoring is enabled.

MQIA_MONITORING_Q
Specifies whether queue monitoring is enabled.

MQIA_OUTBOUND_PORT_MAX
Maximum value in the range for the binding of outgoing channels. MQIA_OUTBOUND_PORT_MAX is valid on z/OS only.

MQIA_OUTBOUND_PORT_MIN
Minimum value in the range for the binding of outgoing channels. MQIA_OUTBOUND_PORT_MIN is valid on z/OS only.

MQIA_PERFORMANCE_EVENT
Control attribute for performance events.

MQIA_PLATFORM
Platform on which the queue manager resides.

MQIA_PUBSUB_CLUSTER
Controls whether this queue manager participates in the publish/subscribe clustering.

MQIA_PUBSUB_MAXMSG_RETRY_COUNT
The number of retries when processing (under sync point) a failed command message

MQIA_PUBSUB_MODE
Inquires if the publish/subscribe engine and the queued publish/subscribe interface are running, which allow applications to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface.

MQIA_PUBSUB_NP_MSG
Specifies whether to discard (or keep) an undelivered input message.

MQIA_PUBSUB_NP_RESP
The behavior of undelivered response messages.

MQIA_PUBSUB_SYNC_PT
Specifies whether only persistent (or all) messages must be processed under sync point.

MQIA_QMGR_CFCONLOS
Specifies action to be taken when the queue manager loses connectivity to the administration structure, or any CF structure with CFCONLOS set to ASQMGR. MQIA_QMGR_CFCONLOS is valid on z/OS only.

MQIA_RECEIVE_TIMEOUT
How long a TCP/IP channel waits to receive data from its partner. MQIA_RECEIVE_TIMEOUT is valid on z/OS only.

MQIA_RECEIVE_TIMEOUT_MIN
Minimum length of time that a TCP/IP channel waits to receive data from its partner. MQIA_RECEIVE_TIMEOUT_MIN is valid on z/OS only.

MQIA_RECEIVE_TIMEOUT_TYPE
Qualifier to apply to the *ReceiveTimeout* parameter. MQIA_RECEIVE_TIMEOUT_TYPE is valid on z/OS only.

MQIA_REMOTE_EVENT
Control attribute for remote events.

MQIA_SECURITY_CASE
Specifies whether the queue manager supports security profile names either in mixed case, or in uppercase only. MQIA_SECURITY_CASE is valid on z/OS only.

MQIA_SHARED_Q_Q_MGR_NAME
When a queue manager makes an MQOPEN call for a shared queue and the queue manager that is specified in the *ObjectQmgrName* parameter of the MQOPEN call is in the same queue-sharing group as the processing queue manager, the SQQMNAME attribute specifies

whether the *ObjectQmgrName* is used or whether the processing queue manager opens the shared queue directly. MQIA_SHARED_Q_Q_MGR_NAME is valid on z/OS only.

MQIA_SSL_EVENT
Control attribute for SSL events.

MQIA_SSL_FIPS_REQUIRED
Specifies whether only FIPS-certified algorithms are to be used if cryptography is executed inIBM WebSphere MQ rather then in the cryptographic hardware itself.

MQIA_SSL_RESET_COUNT
SSL key reset count.

MQIA_SSL_TASKS
SSL tasks. This parameter is valid on z/OS only.

MQIA_START_STOP_EVENT
Control attribute for start stop events.

MQIA_STATISTICS_AUTO_CLUSSDR
Specifies whether statistics data is to be collected for auto-defined cluster-sender channels and, if so, the rate of data collection. MQIA_STATISTICS_AUTO_CLUSSDR is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIA_STATISTICS_CHANNEL
Specifies whether statistics monitoring data is to be collected for channels and, if so, the rate of data collection. MQIA_STATISTICS_CHANNEL is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIA_STATISTICS_INTERVAL
Statistics data collection interval. MQIA_STATISTICS_INTERVAL is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIA_STATISTICS_MQI
Specifies whether statistics monitoring data is to be collected for the queue manager. MQIA_STATISTICS_MQI is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIA_STATISTICS_Q
Specifies whether statistics monitoring data is to be collected for queues. MQIA_STATISTICS_Q is valid only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIA_SUITE_B_STRENGTH
Specifies whether Suite B-compliant cryptography is used and the level of strength employed. For more information about Suite B configuration and its effect on SSL and TLS channels, see NSA Suite B Cryptography in IBM WebSphere MQ .

MQIA_SYNCPOINT
Sync point availability.

MQIA_TCP_CHANNELS
Maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol This is valid on z/OS only.

MQIA_TCP_KEEP_ALIVE
Specifies whether the TCP KEEPALIVE facility is to be used to check whether the other end of a connection is still available. MQIA_TCP_KEEP_ALIVE is valid on z/OS only.

MQIA_TCP_STACK_TYPE
Specifies whether the channel initiator can use only the TCP/IP address space specified in the *TCPName* parameter, or can optionally bind to any selected TCP/IP address. MQIA_TCP_STACK_TYPE is valid on z/OS only.

MQIA_TRACE_ROUTE_RECORDING
Specifies whether trace-route information can be recorded and reply messages generated.

MQIA_TREE_LIFE_TIME
The lifetime of non-administrative topics.

MQIA_TRIGGER_INTERVAL
Trigger interval.

MQIA_XR_CAPABILITY
Specifies whether telemetry commands are supported.

MQIACF_Q_MGR_CLUSTER
All clustering attributes. These attributes are:

- MQCA_CLUSTER_WORKLOAD_DATA
- MQCA_CLUSTER_WORKLOAD_EXIT
- MQCA_CHANNEL_AUTO_DEF_EXIT
- MQCA_REPOSITORY_NAME
- MQCA_REPOSITORY_NAMELIST
- MQIA_CLUSTER_WORKLOAD_LENGTH
- MQIA_CLWL_MRU_CHANNELS
- MQIA_CLWL_USEQ
- MQIA_MONITORING_AUTO_CLUSSDR
- MQCA_Q_MGR_IDENTIFIER

MQIACF_Q_MGR_DQM
All distributed queuing attributes. These attributes are:

- MQCA_CHANNEL_AUTO_DEF_EXIT
- MQCA_DEAD_LETTER_Q_NAME
- MQCA_DEF_XMIT_Q_NAME
- MQCA_DNS_GROUP
- MQCA_IGQ_USER_ID
- MQCA_LU_GROUP_NAME
- MQCA_LU_NAME
- MQCA_LU62_ARM_SUFFIX
- MQCA_Q_MGR_IDENTIFIER
- MQCA_SSL_CRL_NAMELIST
- MQCA_SSL_CRYPTO_HARDWARE
- MQCA_SSL_KEY_REPOSITORY
- MQCA_TCP_NAME
- MQIA_ACTIVE_CHANNELS
- MQIA_ADOPTNEWMCA_CHECK
- MQIA_ADOPTNEWMCA_TYPE
- MQIA_CHANNEL_AUTO_DEF
- MQIA_CHANNEL_AUTO_DEF_EVENT
- MQIA_CHANNEL_EVENT
- MQIA_CHINIT_ADAPTERS
- MQIA_CHINIT_CONTROL
- MQIA_CHINIT_DISPATCHERS
- MQIA_CHINIT_SERVICE_PARM
- MQIA_CHINIT_TRACE_AUTO_START
- MQIA_CHINIT_TRACE_TABLE_SIZE

- MQIA_CHLAUTH_RECORDS
- MQIA_INTRA_GROUP_QUEUING
- MQIA_IGQ_PUT_AUTHORITY
- MQIA_IP_ADDRESS_VERSION
- MQIA_LISTENER_TIMER
- MQIA_LU62_CHANNELS
- MQIA_MAX_CHANNELS
- MQIA_MONITORING_CHANNEL
- MQIA_OUTBOUND_PORT_MAX
- MQIA_OUTBOUND_PORT_MIN
- MQIA_RECEIVE_TIMEOUT
- MQIA_RECEIVE_TIMEOUT_MIN
- MQIA_RECEIVE_TIMEOUT_TYPE
- MQIA_SSL_EVENT
- MQIA_SSL_FIPS_REQUIRED
- MQIA_SSL_RESET_COUNT
- MQIA_SSL_TASKS
- MQIA_STATISTICS_AUTO_CLUSSDR
- MQIA_TCP_CHANNELS
- MQIA_TCP_KEEP_ALIVE
- MQIA_TCP_STACK_TYPE

MQIACF_Q_MGR_EVENT

All event control attributes. These attributes are:

- MQIA_AUTHORITY_EVENT
- MQIA_BRIDGE_EVENT
- MQIA_CHANNEL_EVENT
- MQIA_COMMAND_EVENT
- MQIA_CONFIGURATION_EVENT
- MQIA_INHIBIT_EVENT
- MQIA_LOCAL_EVENT
- MQIA_LOGGER_EVENT
- MQIA_PERFORMANCE_EVENT
- MQIA_REMOTE_EVENT
- MQIA_SSL_EVENT
- MQIA_START_STOP_EVENT

MQIACF_Q_MGR_PUBSUB

All queue manager publish/subscribe attributes. These attributes are:

- MQCA_PARENT
- MQIA_PUBSUB_MAXMSG_RETRY_COUNT
- MQIA_PUBSUB_MODE
- MQIA_PUBSUB_NP_MSG
- MQIA_PUBSUB_NP_RESP
- MQIA_PUBSUB_SYNC_PT
- MQIA_TREE_LIFE_TIME

MQIACF_Q_MGR_SYSTEM

All queue manager system attributes. These attributes are:

- MQCA_COMMAND_INPUT_Q_NAME
- MQCA_CUSTOM
- MQCA_DEAD_LETTER_Q_NAME
- MQCA_Q_MGR_NAME
- MQCA_QSG_NAME
- MQCA_VERSION
- MQIA_ACCOUNTING_CONN_OVERRIDE
- MQIA_ACCOUNTING_INTERVAL
- MQIA_ACCOUNTING_Q
- MQIA_ACTIVITY_CONN_OVERRIDE
- MQIA_ACTIVITY_RECORDING
- MQIA_ACTIVITY_TRACE
- MQCA_ALTERATION_DATE
- MQCA_ALTERATION_TIME
- MQIA_CMD_SERVER_CONTROL
- MQIA_CODED_CHAR_SET_ID
- MQIA_COMMAND_LEVEL
- MQIA_CPI_LEVEL
- MQIA_DIST_LISTS
- MQIA_EXPIRY_INTERVAL
- MQIA_MAX_HANDLES
- MQIA_MAX_MSG_LENGTH
- MQIA_MAX_PRIORITY
- MQIA_MAX_PROPERTIES_LENGTH
- MQIA_MAX_UNCOMMITTED_MSGS
- MQIA_MONITORING_Q
- MQIA_PLATFORM
- MQIA_SHARED_Q_Q_MGR_NAME
- MQIA_STATISTICS_INTERVAL
- MQIA_STATISTICS_MQI
- MQIA_STATISTICS_Q
- MQIA_SYNCPOINT
- MQIA_TRACE_ROUTE_RECORDING
- MQIA_TRIGGER_INTERVAL
- MQIA_XR_CAPABILITY

Inquire Queue Manager (Response):

The response to the Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) command consists of the response header followed by the *QMgrName* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
✓	✓	✓

Always returned:
QMgrName

Returned if requested:
AccountingConnOverride, *AccountingInterval*, *ActivityConnOverride*, *ActivityRecording*, *ActivityTrace*, *AdoptNewMCACheck*, *AdoptNewMCAType*, *AlterationDate*, *AlterationTime*, *AuthorityEvent*, *BridgeEvent*, *CertificateValPolicy*, *CFConlos*, *ChannelAutoDef*, *ChannelAutoDefEvent*, *ChannelAutoDefExit*, *ChannelAuthenticationRecords*, *ChannelEvent*, *ChannelInitiatorControl*, *ChannelMonitoring*, *ChannelStatistics*, *ChinitAdapters*, *ChinitDispatchers*, *ChinitServiceParm*, *ChinitTraceAutoStart*, *ChinitTraceTableSize*, *ClusterSenderMonitoringDefault*, *ClusterSenderStatistics*, *ClusterWorkloadData*, *ClusterWorkloadExit*, *ClusterWorkloadLength*, *CLWLMRUChannels*, *CLWLUseQ*, *CodedCharSetId*, *CommandEvent*, *CommandInputQName*, *CommandLevel*, *CommandServerControl*, *ConfigurationEvent*, *CreationDate*, *CreationTime*, *Custom*, *DeadLetterQName*, *DefClusterXmitQueueType*, *DefXmitQName*, *DistLists*, *DNSGroup*, *DNSWLM*, *EncryptionPolicySuiteB*, *ExpiryInterval*, *GroupUR*, *IGQPutAuthority*, *IGQUserId*, *InhibitEvent*, *IntraGroupQueuing*, *IPAddressVersion*, *ListenerTimer*, *LocalEvent*, *LoggerEvent*, *LUGroupName*, *LUName*, *LU62ARMSuffix*, *LU62Channels*, *MaxChannels*, *MaxActiveChanels*, *MaxHandles*, *MaxMsgLength*, *MaxPriority*, *MaxPropertiesLength*, *MaxUncommittedMsgs*, *MQIAccounting*, *MQIStatisticsOutboundPortMax*, *OutboundPortMin*, *Parent*, *PerformanceEvent*, *Platform*, *PubSubClus*, *PubSubMaxMsgRetryCount*, *PubSubMode*, *QmgrDesc*, *QMgrIdentifier*, *QSGName*, *QueueAccounting*, *QueueMonitoring*, *QueueStatistics*, *ReceiveTimeout*, *ReceiveTimeoutMin*, *ReceiveTimeoutType*, *RemoteEvent*, *RepositoryName*, *RepositoryNamelist*, *SecurityCase*, *SharedQQmgrName*, *Splcap*, *SSLCRLNamelist*, *SSLCryptoHardware*, *SSLEvent*, *SSLFIPSRequired*, *SSLKeyRepository*, *SSLKeyResetCount*, *SSLTasks*, *StartStopEvent*, *StatisticsInterval*, *SyncPoint*, *TCPChannels*, *TCPKeepAlive*, *TCPName*, *TCPStackType*, *TraceRouteRecording*, *TreeLifeTime*, *TriggerInterval*, *Version*

Response data

AccountingConnOverride **(MQCFIN)**
Specifies whether applications can override the settings of the *QueueAccounting* and *MQIAccounting* queue manager parameters (parameter identifier: MQIA_ACCOUNTING_CONN_OVERRIDE).

The value can be:

MQMON_DISABLED
Applications cannot override the settings of the *QueueAccounting* and *MQIAccounting* parameters.

MQMON_ENABLED
Applications can override the settings of the *QueueAccounting* and *MQIAccounting* parameters by using the options field of the MQCNO structure of the MQCONNX API call.

This parameter applies only to AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

AccountingInterval **(MQCFIN)**
The time interval, in seconds, at which intermediate accounting records are written (parameter identifier: MQIA_ACCOUNTING_INTERVAL).

It is a value in the range 1 through 604 000.

This parameter applies only to AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

ActivityConnOverride **(MQCFIN)**
Specifies whether applications can override the setting of the ACTVTRC value in the queue manager attribute (parameter identifier: MQIA_ACTIVITY_CONN_OVERRIDE).

The value can be:

MQMON_DISABLED
Applications cannot override the setting of the ACTVTRC queue manager attribute using the Options field in the MQCNO structure on the MQCONNX call. This is the default value.

MQMON_ENABLED
Applications can override the ACTVTRC queue manager attribute using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This parameter applies only to IBM i, Unix systems, and Windows.

ActivityRecording **(MQCFIN)**
Whether activity reports can be generated (parameter identifier: MQIA_ACTIVITY_RECORDING).

The value can be:

MQRECORDING_DISABLED
Activity reports cannot be generated.

MQRECORDING_MSG
Activity reports can be generated and sent to the destination specified by the originator of the message causing the report.

MQRECORDING_Q
Activity reports can be generated and sent to SYSTEM.ADMIN.ACTIVITY.QUEUE.

ActivityTrace **(MQCFIN)**
Whether activity reports can be generated (parameter identifier: MQIA_ACTIVITY_TRACE).

The value can be:

MQMON_OFF
Do not collect WebSphere MQ MQI application activity trace. This is the default value.

If you set the queue manager attribute ACTVCONO to ENABLED, this value might be overridden for individual connections using the Options field in the MQCNO structure.

MQMON_ON
Collect WebSphere MQ MQI application activity trace.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This parameter applies only to IBM i, Unix systems, and Windows.

AdoptNewMCACheck **(MQCFIN)**
The elements checked to determine whether an MCA must be adopted (restarted) when a new inbound channel is detected. It is adopted if it has the same name as a currently active MCA (parameter identifier: MQIA_ADOPTNEWMCA_CHECK).

The value can be:

MQADOPT_CHECK_Q_MGR_NAME
Check the queue manager name.

MQADOPT_CHECK_NET_ADDR
Check the network address.

MQADOPT_CHECK_ALL
Check the queue manager name and network address.

MQADOPT_CHECK_NONE
Do not check any elements.

This parameter is valid only on z/OS.

AdoptNewMCAType **(MQCFIL)**
Adoption of orphaned channel instances (parameter identifier: MQIA_ADOPTNEWMCA_TYPE).

The value can be:

MQADOPT_TYPE_NO
Do not adopt orphaned channel instances.

MQADOPT_TYPE_ALL
Adopt all channel types.

This parameter is valid only on z/OS.

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date, in the form yyyy-mm-dd, on which the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time, in the form hh.mm.ss, at which the information was last altered.

AuthorityEvent **(MQCFIN)**
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

BridgeEvent **(MQCFIN)**
Controls whether IMS Bridge events are generated (parameter identifier: MQIA_BRIDGE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

This parameter is valid only on z/OS.

CertificateValPolicy **(MQCFIN)**
Specifies which SSL/TLS certificate validation policy is used to validate digital certificates received from remote partner systems (parameter identifier: MQIA_CERT_VAL_POLICY).

This attribute can be used to control how strictly the certificate chain validation conforms to industry security standards. This parameter is only valid on UNIX, Linux, and Windows. For more information, see Certificate validation policies in WebSphere MQ.

The value can be:

MQ_CERT_VAL_POLICY_ANY
Apply each of the certificate validation policies supported by the secure sockets library and accept the certificate chain if any of the policies considers the certificate chain valid. This setting can be used for maximum backwards compatibility with older digital certificates which do not comply with the modern certificate standards.

MQ_CERT_VAL_POLICY_RFC5280
Apply only the RFC 5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

CFConlos **(MQCFIN)**
Specifies the action to be taken when the queue manager loses connectivity to the administration structure, or any CF structures with CFCONLOS set to ASQMGR (parameter identifier: MQIA_QMGR_CFCONLOS).

The value can be:

MQCFCONLOS_TERMINATE
The queue manager terminates when connectivity to CF structures is lost.

MQCFCONLOS_TOLERATE
The queue manager tolerates loss of connectivity to CF structures without terminating.

This parameter is valid only on z/OS.

ChannelAutoDef **(MQCFIN)**
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: MQIA_CHANNEL_AUTO_DEF).

The value can be:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

ChannelAutoDefEvent **(MQCFIN)**
Controls whether channel auto-definition events are generated (parameter identifier: MQIA_CHANNEL_AUTO_DEF_EVENT), when a receiver, server-connection, or cluster-sender channel is auto-defined.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDefExit **(MQCFST)**
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

ChannelAuthenticationRecords **(MQCFIN)**
Controls whether channel authentication records are checked (parameter identifier: MQIA_CHLAUTH_RECORDS).

The value can be:

MQCHLA_DISABLED
Channel authentication records are not checked.

MQCHLA_ENABLED
Channel authentication records are checked.

ChannelEvent **(MQCFIN)**
Controls whether channel events are generated (parameter identifier: MQIA_CHANNEL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_EXCEPTION
Reporting of exception channel events enabled.

ChannelInitiatorControl **(MQCFIN)**
Start the channel initiator during queue manager start (parameter identifier: MQIA_CHINIT_CONTROL). This parameter is not available on z/OS.

The value can be:

MQSVC_CONTROL_MANUAL
The channel initiator is not to be started automatically when the queue manager starts.

MQSVC_CONTROL_Q_MGR
The channel initiator is to be started automatically when the queue manager starts.

ChannelMonitoring **(MQCFIN)**
Default setting for online monitoring for channels (parameter identifier: MQIA_MONITORING_CHANNEL).

If the *ChannelMonitoring* channel attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be:

MQMON_OFF
Online monitoring data collection is turned off.

MQMON_NONE
Online monitoring data collection is turned off for channels regardless of the setting of their *ChannelMonitoring* attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

ChannelStatistics **(MQCFIN)**
Specifies whether statistics data is to be collected for channels (parameter identifier: MQIA_STATISTICS_CHANNEL).

The value can be:

MQMON_NONE
Statistics data collection is turned off for channels regardless of the setting of their *ChannelStatistics* parameter. MQMON_NONE is the initial default value of the queue manager.

MQMON_OFF
Statistics data collection is turned off for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

MQMON_LOW
Statistics data collection is turned on, with a low ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

MQMON_MEDIUM
Statistics data collection is turned on, with a moderate ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

MQMON_HIGH
Statistics data collection is turned on, with a high ratio of data collection, for channels specifying a value of MQMON_Q_MGR in their *ChannelStatistics* parameter.

This parameter applies only to AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

ChinitAdapters **(MQCFIN)**
Number of adapter subtasks (parameter identifier: MQIA_CHINIT_ADAPTERS).

The number of adapter subtasks to use for processing WebSphere MQ calls. This parameter is valid only on z/OS.

ChinitDispatchers **(MQCFIN)**
Number of dispatchers (parameter identifier: MQIA_CHINIT_DISPATCHERS).

The number of dispatchers to use for the channel initiator. This parameter is valid only on z/OS.

ChinitServiceParm **(MQCFST)**
Reserved for use by IBM (parameter identifier: MQCA_CHINIT_SERVICE_PARM).

ChinitTraceAutoStart **(MQCFIN)**
Specifies whether the channel initiator trace must start automatically (parameter identifier: MQIA_CHINIT_TRACE_AUTO_START).

The value can be:

MQTRAXSTR_YES
Channel initiator trace is to start automatically.

MQTRAXSTR_NO
Channel initiator trace is not to start automatically.

This parameter is valid only on z/OS.

ChinitTraceTableSize **(MQCFIN)**
The size, in megabytes, of the trace data space of the channel initiator (parameter identifier: MQIA_CHINIT_TRACE_TABLE_SIZE).

This parameter is valid only on z/OS.

ClusterSenderMonitoringDefault **(MQCFIN)**
Setting for online monitoring for automatically defined cluster-sender channels (parameter identifier: MQIA_MONITORING_AUTO_CLUSSDR).

The value can be:

MQMON_Q_MGR
Collection of online monitoring data is inherited from the setting of the queue manager's *ChannelMonitoring* parameter.

MQMON_OFF
Monitoring for the channel is switched off.

MQMON_LOW
Specifies a low rate of data collection with a minimal effect on system performance unless *ChannelMonitoring* for the queue manager is MQMON_NONE. The data collected is not likely to be the most current.

MQMON_MEDIUM
Specifies a moderate rate of data collection with limited effect on system performance unless *ChannelMonitoring* for the queue manager is MQMON_NONE.

MQMON_HIGH
Specifies a high rate of data collection with a likely effect on system performance unless *ChannelMonitoring* for the queue manager is MQMON_NONE. The data collected is the most current available.

ClusterSenderStatistics **(MQCFIN)**
Specifies whether statistics data is to be collected for auto-defined cluster-sender channels (parameter identifier: MQIA_STATISTICS_AUTO_CLUSSDR).

The value can be:

MQMON_Q_MGR
Collection of statistics data is inherited from the setting of the queue manager's *ChannelStatistics* parameter.

MQMON_OFF
Statistics data collection for the channel is switched off.

MQMON_LOW
Specifies a low rate of data collection with a minimal effect on system performance.

MQMON_MEDIUM
Specifies a moderate rate of data collection.

MQMON_HIGH
Specifies a high rate of data collection.

This parameter applies only to AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

ClusterWorkLoadData **(MQCFST)**
Data passed to the cluster workload exit (parameter identifier: MQCA_CLUSTER_WORKLOAD_DATA).

ClusterWorkLoadExit **(MQCFST)**
Name of the cluster workload exit (parameter identifier: MQCA_CLUSTER_WORKLOAD_EXIT).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

ClusterWorkLoadLength **(MQCFIN)**
Cluster workload length (parameter identifier: MQIA_CLUSTER_WORKLOAD_LENGTH).

The maximum length of the message passed to the cluster workload exit.

CLWLMRUChannels **(MQCFIN)**
Cluster workload most recently used (MRU) channels (parameter identifier: MQIA_CLWL_MRU_CHANNELS).

The maximum number of active most recently used outbound channels.

CLWLUseQ **(MQCFIN)**
Use of remote queue (parameter identifier: MQIA_CLWL_USEQ).

Specifies whether a cluster queue manager is to use remote puts to other queues defined in other queue managers within the cluster during workload management.

The value can be:

MQCLWL_USEQ_ANY
Use remote queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CodedCharSetId **(MQCFIN)**
Coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

CommandEvent **(MQCFIN)**
Controls whether command events are generated (parameter identifier: MQIA_COMMAND_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MQEVR_NODISPLAY
Event reporting enabled for all successful commands except Inquire commands.

CommandInputQName **(MQCFST)**
Command input queue name (parameter identifier: MQCA_COMMAND_INPUT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

CommandLevel **(MQCFIN)**
Command level supported by queue manager (parameter identifier: MQIA_COMMAND_LEVEL).

The value can be:

MQCMDL_LEVEL_1
Level 1 of system control commands.

This value is returned by the following platforms:
- MQSeries for AIX V2.2
- MQSeries for OS/400®:
  - V2R3
  - V3R1
  - V3R6
- MQSeries for Windows V2.0

MQCMDL_LEVEL_101
MQSeries for Windows V2.0.1

MQCMDL_LEVEL_110
MQSeries for Windows V2.1

MQCMDL_LEVEL_200
MQSeries for Windows NT V2.0

MQCMDL_LEVEL_220
Level 220 of system control commands.

This value is returned by the following platforms:
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for Compaq NonStop Kernel V2.2.0.1

MQCMDL_LEVEL_221
Level 221 of system control commands.

This value is returned by the following platforms:
- MQSeries for AIX Version 2.2.1
- MQSeries for DIGITAL UNIX (Compaq Tru64 UNIX) V2.2.1

MQCMDL_LEVEL_320
MQSeries for OS/400 V3R2 and V3R7

MQCMDL_LEVEL_420
MQSeries for AS/400 V4R2 and R2.1

MQCMDL_LEVEL_500
Level 500 of system control commands.

This value is returned by the following platforms:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for Solaris V5.0
- MQSeries for Windows NT V5.0

MQCMDL_LEVEL_510
Level 510 of system control commands.

This value is returned by the following platforms:

- MQSeries for AIX V5.1
- MQSeries for AS/400 V5.1
- MQSeries for HP-UX V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- IBM WebSphere MQ for HP Integrity NonStop Server v5.3
- MQSeries for Solaris V5.1
- MQSeries for Windows NT V5.1

MQCMDL_LEVEL_520
Level 520 of system control commands.

This value is returned by the following platforms:

- MQSeries for AIX V5.2
- MQSeries for AS/400 V5.2
- MQSeries for HP-UX V5.2
- MQSeries for Linux V5.2
- MQSeries for Solaris V5.2
- MQSeries for Windows NT V5.2
- MQSeries for Windows 2000 V5.2

MQCMDL_LEVEL_530
Level 530 of system control commands.

This value is returned by the following platforms:

- WebSphere MQ for AIX, V5.3
- WebSphere MQ for IBM i, V5.3
- WebSphere MQ for HP-UX, V5.3
- WebSphere MQ for Linux, V5.3
- WebSphere MQ for Sun Solaris, Version 5.3
- WebSphere MQ for Windows NT and Windows 2000, Version 5.3

MQCMDL_LEVEL_531
Level 531 of system control commands.

MQCMDL_LEVEL_600
Level 600 of system control commands.

MQCMDL_LEVEL_700
Level 700 of system control commands.

MQCMDL_LEVEL_701
Level 701 of system control commands.

MQCMDL_LEVEL_710
Level 710 of system control commands.

The set of system control commands that corresponds to a particular value of the *CommandLevel* attribute varies. It varies according to the value of the *Platform* attribute; both must be used to decide which system control commands are supported.

CommandServerControl **(MQCFIN)**
Start the command server during queue manager start (parameter identifier: MQIA_CMD_SERVER_CONTROL). This parameter is not available on z/OS.

The value can be:

MQSVC_CONTROL_MANUAL
The command server is not to be started automatically when the queue manager starts.

MQSVC_CONTROL_Q_MGR
The command server is to be started automatically when the queue manager starts.

ConfigurationEvent **(MQCFIN)**
Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

CreationDate **(MQCFST)**

Queue creation date, in the form yyyy-mm-dd (parameter identifier: MQCA_CREATION_DATE). The maximum length of the string is MQ_CREATION_DATE_LENGTH.

CreationTime **(MQCFST)**

Creation time, in the form hh.mm.ss (parameter identifier: MQCA_CREATION_TIME). The maximum length of the string is MQ_CREATION_TIME_LENGTH.

Custom **(MQCFST)**
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes are introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form NAME(VALUE).

This description is updated when features using this attribute are introduced.

DeadLetterQName **(MQCFST)**
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DefClusterXmitQueueType **(MQCFIN)**
The DefClusterXmitQueueType attribute controls which transmission queue is selected by default by

cluster-sender channels to get messages from, to send the messages to cluster-receiver channels. (Parameter identifier: MQIA_DEF_CLUSTER_XMIT_Q_TYPE.)

The values of DefClusterXmitQueueType are MQCLXQ_SCTQ or MQCLXQ_CHANNEL.

MQCLXQ_SCTQ

All cluster-sender channels send messages from SYSTEM.CLUSTER.TRANSMIT.QUEUE. The correlID of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute DefClusterXmitQueueType was not present.

MQCLXQ_CHANNEL
Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

The attribute is not supported on z/OS.

DefXmitQName **(MQCFST)**
Default transmission queue name (parameter identifier: MQCA_DEF_XMIT_Q_NAME).

The default transmission queue is used for the transmission of messages to remote queue managers. It is used if there is no other indication of which transmission queue to use.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DistLists **(MQCFIN)**
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

The value can be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

DNSGroup **(MQCFST)**
DNS group name (parameter identifier: MQCA_DNS_GROUP).

The name of the group that the TCP listener handling inbound transmissions for the queue-sharing group joins. It must join this group when using Workload Manager for Dynamic Domain Name Services support (DDNS).

This parameter is valid only on z/OS.

DNSWLM **(MQCFIN)**
Controls whether the TCP listener that handles inbound transmissions for the queue-sharing group must register with Workload Manager (WLM) for DDNS: (parameter identifier: MQIA_DNS_WLM).

The value can be:

MQDNSWLM_YES
The listener must register with WLM.

MQDNSWLM_NO
The listener is not to register with WLM. MQDNSWLM_NO is the initial default value of the queue manager.

This parameter is valid only on z/OS.

EncryptionPolicySuiteB **(MQCFIL)**
Specifies whether Suite B-compliant cryptography is used and what level of strength is employed (parameter identifier: MQIA_SUITE_B_STRENGTH). For more information about Suite B configuration and its effect on SSL and TLS channels, see NSA Suite B Cryptography in IBM WebSphere MQ.

The value can be one, or more, of:

MQ_SUITE_B_NONE
Suite B-compliant cryptography is not used.

MQ_SUITE_B_128_BIT
Suite B 128-bit strength security is used.

MQ_SUITE_B_192_BIT
Suite B 192-bit strength security is used.

MQ_SUITE_B_128_BIT,MQ_SUITE_B_192_BIT
Suite B 128-bit and Suite B 192-bit strength security is used.

ExpiryInterval **(MQCFIN)**
Interval between scans for expired messages (parameter identifier: MQIA_EXPIRY_INTERVAL).

Specifies the frequency with which the queue manager scans the queues looking for expired messages. This parameter is a time interval in seconds in the range 1 through 99 999 999, or the following special value:

MQEXPI_OFF
No scans for expired messages.

This parameter is valid only on z/OS.

GroupUR **(MQCFIN)**
Identifies whether XA client applications can establish transactions with a GROUP unit of recovery disposition.

The value can be:

MQGUR_DISABLED
XA client applications must connect using a queue manager name.

MQGUR_ENABLED
XA client applications can establish transactions with a group unit of recovery disposition by specifying a QSG name when they connect.

This parameter is valid only on z/OS.

IGQPutAuthority **(MQCFIN)**
Type of authority checking used by the intra-group queuing agent (parameter identifier: MQIA_IGQ_PUT_AUTHORITY).

The attribute indicates the type of authority checking that is performed by the local intra-group queuing agent (IGQ agent). The checking is performed when the IGQ agent removes a message from the shared transmission queue and places the message on a local queue. The value can be:

MQIGQPA_DEFAULT
Default user identifier is used.

MQIGQPA_CONTEXT
Context user identifier is used.

MQIGQPA_ONLY_IGQ
Only the IGQ user identifier is used.

MQIGQPA_ALTERNATE_OR_IGQ
Alternate user identifier or IGQ-agent user identifier is used.

This parameter is valid only on z/OS.

IGQUserId **(MQCFST)**
User identifier used by the intra-group queuing agent (parameter identifier: MQCA_IGQ_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH. This parameter is valid only on z/OS.

InhibitEvent **(MQCFIN)**
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

IntraGroupQueuing **(MQCFIN)**
Specifies whether intra-group queuing is used (parameter identifier: MQIA_INTRA_GROUP_QUEUING).

The value can be:

MQIGQ_DISABLED
Intra-group queuing is disabled. All messages destined for other queue managers in the queue-sharing group are transmitted using conventional channels.

MQIGQ_ENABLED
Intra-group queuing is enabled.

This parameter is valid only on z/OS.

IPAddressVersion **(MQCFIN)**
IP address version selector (parameter identifier: MQIA_IP_ADDRESS_VERSION).

Specifies which IP address version, either IPv4 or IPv6, is used. The value can be:

MQIPADDR_IPV4
IPv4 is used.

MQIPADDR_IPV6
IPv6 is used.

ListenerTimer **(MQCFIN)**
Listener restart interval (parameter identifier: MQIA_LISTENER_TIMER).

The time interval, in seconds, between attempts by WebSphere MQ to restart the listener after an APPC or TCP/IP failure.

LocalEvent **(MQCFIN)**
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

This parameter is valid only on z/OS.

LoggerEvent **(MQCFIN)**
Controls whether recovery log events are generated (parameter identifier: MQIA_LOGGER_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

This parameter applies only to AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

LUGroupName **(MQCFST)**
Generic LU name for the LU 6.2 listener (parameter identifier: MQCA_LU_GROUP_NAME).

The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group. This parameter is valid only on z/OS.

LUName **(MQCFST)**
LU name to use for outbound LU 6.2 transmissions (parameter identifier: MQCA_LU_NAME).

The name of the LU to use for outbound LU 6.2 transmissions. This parameter is valid only on z/OS.

LU62ARMSuffix **(MQCFST)**
APPCPM suffix (parameter identifier: MQCA_LU62_ARM_SUFFIX).

The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. This parameter is valid only on z/OS.

LU62Channels **(MQCFIN)**
Maximum number of LU 6.2 channels (parameter identifier: MQIA_LU62_CHANNELS).

The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol. This parameter is valid only on z/OS.

MaxActiveChannels **(MQCFIN)**
Maximum number of channels (parameter identifier: MQIA_ACTIVE_CHANNELS).

The maximum number of channels that can be active at any time. This parameter is valid only on z/OS.

MaxChannels **(MQCFIN)**
Maximum number of current channels (parameter identifier: MQIA_MAX_CHANNELS).

The maximum number of channels that can be current (including server-connection channels with connected clients). This parameter is valid only on z/OS.

MaxHandles **(MQCFIN)**
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

Specifies the maximum number of handles that any one connection can have open at the same time.

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

MaxPriority **(MQCFIN)**
Maximum priority (parameter identifier: MQIA_MAX_PRIORITY).

MaxPropertiesLength **(MQCFIN)**
Maximum properties length (parameter identifier: MQIA_MAX_PROPERTIES_LENGTH).

MaxUncommittedMsgs **(MQCFIN)**
Maximum number of uncommitted messages within a unit of work (parameter identifier: MQIA_MAX_UNCOMMITTED_MSGS).

This number is the sum of the following number of messages under any one sync point. :

- The number of messages that can be retrieved, plus
- The number of messages that can be put on a queue, plus
- Any trigger messages generated within this unit of work

The limit does not apply to messages that are retrieved or put outside sync point.

MQIAccounting **(MQCFIN)**
Specifies whether accounting information for MQI data is to be collected (parameter identifier: MQIA_ACCOUNTING_MQI).

The value can be:

MQMON_OFF
MQI accounting data collection is disabled.

MQMON_ON
MQI accounting data collection is enabled.

This parameter applies only to AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MQIStatistics **(MQCFIN)**
Specifies whether statistics monitoring data is to be collected for the queue manager (parameter identifier: MQIA_STATISTICS_MQI).

The value can be:

MQMON_OFF
Data collection for MQI statistics is disabled. MQMON_OFF is the initial default value of the queue manager.

MQMON_ON
Data collection for MQI statistics is enabled.

This parameter applies only to AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

MsgMarkBrowseInterval (MQCFIN)
Mark-browse interval (parameter identifier: MQIA_MSG_MARK_BROWSE_INTERVAL).

The time interval in milliseconds after which the queue manager can automatically unmark messages.

OutboundPortMax **(MQCFIN)**
The maximum value in the range for the binding of outgoing channels (parameter identifier: MQIA_OUTBOUND_PORT_MAX).

The maximum value in the range of port numbers to be used when binding outgoing channels. This parameter is valid only on z/OS.

OutboundPortMin **(MQCFIN)**
The minimum value in the range for the binding of outgoing channels (parameter identifier: MQIA_OUTBOUND_PORT_MIN).

The minimum value in the range of port numbers to be used when binding outgoing channels. This parameter is valid only on z/OS.

Parent **(MQCFST)**
The name of the hierarchically connected queue manager nominated as the parent of this queue manager (parameter identifier: MQCA_PARENT).

PerformanceEvent **(MQCFIN)**
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Platform **(MQCFIN)**
Platform on which the queue manager resides (parameter identifier: MQIA_PLATFORM).

The value can be:

MQPL_AIX
AIX (same value as MQPL_UNIX).

MQPL_NSK
HP Integrity NonStop Server.

MQPL_OS400
IBM i.

MQPL_UNIX
UNIX systems.

MQPL_WINDOWS_NT
Windows.

MQPL_ZOS
z/OS

PubSubClus **(MQCFIN)**
Controls whether the queue manager participates in publish/subscribe clustering (parameter identifier: MQIA_PUBSUB_CLUSTER).

The value can be:

MQPSCLUS_ENABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is permitted.

Note: The introduction of a clustered topic into a large IBM WebSphere MQ cluster can cause a degradation in performance. This degradation occurs because all partial repositories are notified of all the other members of the cluster. Unexpected subscriptions might be created at all other nodes; for example, where proxysub(FORCE) is specified. Large numbers of channels might be started from a queue manager; for example, on resync after a queue manager failure.

MQPSCLUS_DISABLED
The creating or receipt of clustered topic definitions and cluster subscriptions is inhibited. The creations or receipts are recorded as warnings in the queue manager error logs.

PubSubMaxMsgRetryCount **(MQCFIN)**
The number of attempts to reprocess a failed command message under sync point (parameter identifier: MQIA_PUBSUB_MAXMSG_RETRY_COUNT).

PubSubMode **(MQCFIN)**
Specifies whether the publish/subscribe engine and the queued publish/subscribe interface are running. The publish/subscribe engine enables applications to publish or subscribe by using the application programming interface. The publish/subscribe interface monitors the queues used the queued publish/subscribe interface (parameter identifier: MQIA_PUBSUB_MODE).

The values can be as follows:

MQPSM_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish or subscribe by using the application programming interface. The queued publish/subscribe interface is not running. Therefore any message that is put to the queues that are monitored by the queued publish/subscribe interface is not acted on. MQPSM_COMPAT is used for compatibility with WebSphere Message Broker V6 or earlier versions of WebSphere Message Broker that use this queue manager. WebSphere Message Broker reads the same queues from which the queued publish/subscribe interface normally reads.

MQPSM_DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish or subscribe by using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted on.

MQPSM_ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish or subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface. MQPSM_ENABLED is the initial default value of the queue manager.

PubSubNPInputMsg **(MQCFIN)**
Specifies whether to discard or keep an undelivered input message (parameter identifier: MQIA_PUBSUB_NP_MSG).

The values can be as follows:

MQUNDELIVERED_DISCARD
Non-persistent input messages can be discarded if they cannot be processed. MQUNDELIVERED_DISCARD is the default value.

MQUNDELIVERED_KEEP
Non-persistent input messages are not discarded if they cannot be processed. The queued publish/subscribe interface continues to try the process again at appropriate intervals. It does not continue processing subsequent messages.

PubSubNPResponse **(MQCFIN)**
Controls the behavior of undelivered response messages (parameter identifier: MQIA_PUBSUB_NP_RESP).

The values can be as follows:

MQUNDELIVERED_NORMAL
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If they cannot be placed on the dead letter queue, they are discarded.

MQUNDELIVERED_SAFE
Non-persistent responses that cannot be placed on the reply queue are put on the dead letter queue. If the response cannot be sent and cannot be placed on the dead letter queue the queued publish/subscribe interface rolls back the current operation. The operation is tried again at appropriate intervals and does not continue processing subsequent messages.

MQUNDELIVERED_DISCARD
Non-persistent responses that cannot be placed on the reply queue are discarded. MQUNDELIVERED_DISCARD is the default value for new queue managers.

MQUNDELIVERED_KEEP
Non-persistent responses are not placed on the dead letter queue or discarded. Instead, the queued publish/subscribe interface backs out the current operation and then tries it again at appropriate intervals.

PubSubSyncPoint **(MQCFIN)**
Specifies whether only persistent messages or all messages are processed under sync point (parameter identifier: MQIA_PUBSUB_SYNC_PT).

The values can be as follows:

MQSYNCPOINT_IFPER
This makes the queued publish/subscribe interface receive non-persistent messages outside sync point. If the daemon receives a publication outside sync point, the daemon forwards the publication to subscribers known to it outside sync point. MQSYNCPOINT_IFPER is the default value.

MQSYNCPOINT_YES
MQSYNCPOINT_YES makes the queued publish/subscribe interface receive all messages under sync point.

QMgrDesc **(MQCFST)**
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

This parameter is text that briefly describes the object.

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager on which the command is executing. Using this character set ensures that the text is translated correctly.

QMgrIdentifier **(MQCFST)**
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

QMgrName **(MQCFST)**
Name of local queue manager (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QSGName **(MQCFST)**
Queue sharing group name (parameter identifier: MQCA_QSG_NAME).

The maximum length of the string is MQ_QSG_NAME_LENGTH. This parameter is valid only on z/OS.

QueueAccounting **(MQCFIN)**
Collection of accounting (thread-level and queue-level accounting) data for queues (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_NONE
Accounting data collection for queues is disabled.

MQMON_OFF
Accounting data collection is disabled for queues specifying a value of MQMON_Q_MGR in the *QueueAccounting* parameter.

MQMON_ON
Accounting data collection is enabled for queues specifying a value of MQMON_Q_MGR in the *QueueAccounting* parameter.

QueueMonitoring **(MQCFIN)**
Default setting for online monitoring for queues (parameter identifier: MQIA_MONITORING_Q).

If the *QueueMonitoring* queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be:

MQMON_OFF
Online monitoring data collection is turned off.

MQMON_NONE
Online monitoring data collection is turned off for queues regardless of the setting of their *QueueMonitoring* attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

QueueStatistics **(MQCFIN)**
Specifies whether statistics data is to be collected for queues (parameter identifier: MQIA_STATISTICS_Q).

The value can be:

MQMON_NONE
Statistics data collection is turned off for queues regardless of the setting of their *QueueStatistics* parameter.

MQMON_OFF
Statistics data collection is turned off for queues specifying a value of MQMON_Q_MGR in their *QueueStatistics* parameter.

MQMON_ON
Statistics data collection is turned on for queues specifying a value of MQMON_Q_MGR in their *QueueStatistics* parameter.

This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

ReceiveTimeout **(MQCFIN)**
How long a TCP/IP channel waits to receive data from its partner (parameter identifier: MQIA_RECEIVE_TIMEOUT).

The length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state.

This parameter is valid only on z/OS.

ReceiveTimeoutMin **(MQCFIN)**
The minimum length of time that a TCP/IP channel waits to receive data from its partner (parameter identifier: MQIA_RECEIVE_TIMEOUT_MIN).

The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. This parameter is valid only on z/OS.

ReceiveTimeoutType **(MQCFIN)**
The qualifier to apply to *ReceiveTimeout* (parameter identifier: MQIA_RECEIVE_TIMEOUT_TYPE).

The qualifier to apply to *ReceiveTimeoutType* to calculate how long a TCP/IP channel waits to receive data from its partner. The wait includes heartbeats. If the wait interval expires the channel returns to the inactive state. This parameter is valid only on z/OS.

The value can be:

MQRCVTIME_MULTIPLY
The *ReceiveTimeout* value is a multiplier to be applied to the negotiated value of *HeartbeatInterval* to determine how long a channel waits.

MQRCVTIME_ADD
ReceiveTimeout is a value, in seconds, to be added to the negotiated value of *HeartbeatInterval* to determine how long a channel waits.

MQRCVTIME_EQUAL
ReceiveTimeout is a value, in seconds, representing how long a channel waits.

RemoteEvent **(MQCFIN)**
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

RepositoryName **(MQCFST)**
Repository name (parameter identifier: MQCA_REPOSITORY_NAME).

The name of a cluster for which this queue manager is to provide a repository service.

RepositoryNamelist **(MQCFST)**
Repository name list (parameter identifier: MQCA_REPOSITORY_NAMELIST).

The name of a list of clusters for which this queue manager is to provide a repository service.

SecurityCase **(MQCFIN)**
Security case supported (parameter identifier: MQIA_SECURITY_CASE).

Specifies whether the queue manager supports security profile names in mixed case, or in uppercase only. The value is activated when a Refresh Security command is run with *SecurityType(MQSECTYPE_CLASSES)* specified.

The value can be:

MQSCYC_UPPER
Security profile names must be in uppercase.

MQSCYC_MIXED
Security profile names can be in uppercase or in mixed case.

This parameter is valid only on z/OS.

SharedQQmgrName **(MQCFIN)**
Shared-queue queue manager name (parameter identifier: MQIA_SHARED_Q_Q_MGR_NAME).

A queue manager makes an MQOPEN call for a shared queue. The queue manager that is specified in the *ObjectQmgrName* parameter of the MQOPEN call is in the same queue-sharing group as the processing queue manager. The SQQMNAME attribute specifies whether the *ObjectQmgrName* is used or whether the processing queue manager opens the shared queue directly.

The value can be:

MQSQQM_USE
ObjectQmgrName is used and the appropriate transmission queue is opened.

MQSQQM_IGNORE
The processing queue manager opens the shared queue directly.

This parameter is valid only on z/OS.

Splcap **(MQCFIN)**
If the WebSphere MQ AMS component is installed for the version of WebSphere MQ that the queue manager is running under, the attribute has a value YES(MQCAP_SUPPORTED). If the WebSphere MQ AMS component is not installed, the value is NO(MQCAP_NOT_SUPPORTED) (parameter identifier: MQIA_PROT_POLICY_CAPABILITY).

The value can be one of the following values:

MQCAP_SUPPORTED
If the WebSphere MQ AMS component is installed for the version of WebSphere MQ that the queue manager is running under.

MQCAP_NOT_SUPPORTED
If the WebSphere MQ AMS component is not installed.

SSLCRLNamelist **(MQCFST)**
The SSL certificate revocation location namelist (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The length of the string is MQ_NAMELIST_NAME_LENGTH.

Indicates the name of a namelist of authentication information objects to be used for certificate revocation checking by the queue manager.

SSLCryptoHardware **(MQCFST)**
Parameters to configure the SSL cryptographic hardware (parameter identifier: MQCA_SSL_CRYPTO_HARDWARE).

The length of the string is MQ_SSL_CRYPTO_HARDWARE_LENGTH.

Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

This parameter is supported on AIX, HP-UX, Solaris, Linux, and Windows only.

SSLEvent **(MQCFIN)**
Controls whether SSL events are generated (parameter identifier: MQIA_SSL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

SSLFipsRequired **(MQCFIN)**
Controls whether only FIPS-certified algorithms are to be used if cryptography is executed in WebSphere MQ itself (parameter identifier: MQIA_SSL_FIPS_REQUIRED). This parameter is valid only on Windows Linux UNIX and z/OS platforms.

The value can be:

MQSSL_FIPS_NO
Any supported CipherSpec can be used.

MQSSL_FIPS_YES
Only FIPS-certified cryptographic algorithms are to be used if cryptography is executed in WebSphere MQ rather than cryptographic hardware.

SSLKeyRepository **(MQCFST)**
Location and name of the SSL key repository (parameter identifier: MQCA_SSL_KEY_REPOSITORY).

The length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

Indicates the name of the Secure Sockets Layer key repository.

The format of the name depends on the environment.

SSLKeyResetCount **(MQCFIN)**
SSL key reset count (parameter identifier: MQIA_SSL_RESET_COUNT).

The number of unencrypted bytes that initiating SSL channel MCAs send or receive before renegotiating the secret key.

SSLTasks **(MQCFIN)**
Number of server subtasks used for processing SSL calls (parameter identifier: MQIA_SSL_TASKS).

The number of server subtasks used for processing SSL calls. This parameter is valid only on z/OS.

StartStopEvent **(MQCFIN)**
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

StatisticsInterval **(MQCFIN)**
The time interval, in seconds, at which statistics monitoring data is written to the monitoring queue (parameter identifier: MQIA_STATISTICS_INTERVAL).

This parameter is valid only on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

SyncPoint **(MQCFIN)**
Sync point availability (parameter identifier: MQIA_SYNCPOINT).

The value can be:

MQSP_AVAILABLE
Units of work and sync pointing available.

MQSP_NOT_AVAILABLE
Units of work and sync pointing not available.

TCPChannels **(MQCFIN)**
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol (parameter identifier: MQIA_TCP_CHANNELS).

This parameter is valid only on z/OS.

TCPKeepAlive **(MQCFIN)**
Specifies whether the TCP KEEPALIVE facility is to be used to check whether the other end of the connection is still available (parameter identifier: MQIA_TCP_KEEP_ALIVE).

The value can be:

MQTCPKEEP_YES
The TCP KEEPALIVE facility is to be used as specified in the TCP profile configuration data set. The interval is specified in the *KeepAliveInterval* channel attribute.

MQTCPKEEP_NO
The TCP KEEPALIVE facility is not to be used.

This parameter is valid only on z/OS.

TCPName **(MQCFST)**
The name of the TCP/IP system that you are using (parameter identifier: MQIA_TCP_NAME).

This parameter is valid only on z/OS.

TCPStackType **(MQCFIN)**
Specifies whether the channel initiator can use only the TCP/IP address space specified in *TCPName*, or can optionally bind to any selected TCP/IP address (parameter identifier: MQIA_TCP_STACK_TYPE).

The value can be:

MQTCPSTACK_SINGLE
The channel initiator can use only the TCP/IP address space specified in *TCPName*.

MQTCPSTACK_MULTIPLE
The channel initiator can use any TCP/IP address space available to it.

This parameter is valid only on z/OS.

TraceRouteRecording **(MQCFIN)**
Specifies whether trace-route information can be recorded and a reply message generated (parameter identifier: MQIA_TRACE_ROUTE_RECORDING).

The value can be:

MQRECORDING_DISABLED
Trace-route information cannot be recorded.

MQRECORDING_MSG
Trace-route information can be recorded and sent to the destination specified by the originator of the message causing the trace route record.

MQRECORDING_Q
Trace-route information can be recorded and sent to SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

TreeLifeTime **(MQCFIN)**
The lifetime in seconds of non-administrative topics (parameter identifier: MQIA_TREE_LIFE_TIME).

Non-administrative topics are those topics created when an application publishes to, or subscribes on, a topic string that does not exist as an administrative node. When this non-administrative node no longer has any active subscriptions, this parameter determines how long the queue manager waits before removing that node. Only non-administrative topics that are in use by a durable subscription remain after the queue manager it recycled.

The value can be in the range 0 - 604,000. A value of 0 means that non-administrative topics are not removed by the queue manager. The initial default value of the queue manager is 1800.

TriggerInterval **(MQCFIN)**
Trigger interval (parameter identifier: MQIA_TRIGGER_INTERVAL).

Specifies the trigger time interval, expressed in milliseconds, for use only with queues where *TriggerType* has a value of MQTT_FIRST.

Version **(MQCFST)**
The version of the IBM WebSphere MQ code (parameter identifier: MQCA_VERSION).

The version of the WebSphere MQ code is shown as VVRRMMFF:

VV: Version

RR: Release

MM: Maintenance level

FF: Fix level

XrCapability **(MQCFIN)**

Specifies whether the IBM WebSphere MQ Telemetry capability and commands are supported by the queue manager where *XrCapability* has a value of MQCAP_SUPPORTED or MQCAP_NOT_SUPPORTED (parameter identifier: MQIA_XR_CAPABILITY).

This parameter applies only to IBM i, Unix systems, and Windows.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

Inquire Queue Manager Status:

The Inquire Queue Manager Status (MQCMD_INQUIRE_Q_MGR_STATUS) command inquires about the status of the local queue manager.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Optional parameters

QMStatusAttrs **(MQCFIL)**
Queue manager status attributes (parameter identifier: MQIACF_Q_MGR_STATUS_ATTRS).

The attribute list might specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_Q_MGR_NAME
Name of the local queue manager.

MQCA_INSTALLATION_DESC
Description of the installation associated with the queue manager. This parameter is not valid on IBM i.

MQCA_INSTALLATION_NAME
Name of the installation associated with the queue manager. This parameter is not valid on IBM i.

MQCA_INSTALLATION_PATH
Path of the installation associated with the queue manager. This parameter is not valid on IBM i.

MQCACF_CURRENT_LOG_EXTENT_NAME
Name of the log extent currently being written to by the logger. MQCACF_CURRENT_LOG_EXTENT_NAME is available only on queue managers using linear logging. On other queue managers, MQCACF_CURRENT_LOG_EXTENT_NAME is blank.

MQCACF_LOG_PATH
Location of the recovery log extents.

MQCACF_MEDIA_LOG_EXTENT_NAME
Name of the earliest log extent required to perform media recovery. MQCACF_MEDIA_LOG_EXTENT_NAME is available only on queue managers using linear logging. On other queue managers, MQCACF_MEDIA_LOG_EXTENT_NAME is blank.

MQCACF_RESTART_LOG_EXTENT_NAME
Name of the earliest log extent required to perform restart recovery. MQCACF_RESTART_LOG_EXTENT_NAME is available only on queue managers using linear logging. On other queue managers, MQCACF_RESTART_LOG_EXTENT_NAME is blank.

MQIACF_CHINIT_STATUS
Current status of the channel initiator.

MQIACF_CMD_SERVER_STATUS
Current status of the command server.

MQIACF_CONNECTION_COUNT
Current number of connections to the queue manager.

MQIACF_Q_MGR_STATUS
Current status of the queue manager.

MQCACF_Q_MGR_START_DATE
The date on which the queue manager was started (in the form yyyy-mm-dd). The length of this attribute is given by MQ_DATE_LENGTH.

MQCACF_Q_MGR_START_TIME
The time at which the queue manager was started (in the form hh.mm.ss). The length of this attribute is given by MQ_TIME_LENGTH.

Inquire Queue Manager Status (Response):

The response to the Inquire Queue Manager Status (MQCMD_INQUIRE_Q_MGR_STATUS) command consists of the response header followed by the *QMgrName* and *QMgrStatus* structures and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned:
QMgrName, *QMgrStatus*

Returned if requested:
ChannelInitiatorStatus, *CommandServerStatus*, *ConnectionCount*, *CurrentLog*, *InstallationDesc*, *InstallationName*, *InstallationPath*, *LogPath*, *MediaRecoveryLog*, *RestartRecoveryLog*, *StartDate*, *StartTime*

Response data

ChannelInitiatorStatus **(MQCFIN)**
Status of the channel initiator reading SYSTEM.CHANNEL.INITQ (parameter identifier: MQIACF_CHINIT_STATUS).

The value can be:

MQSVC_STATUS_STOPPED
The channel initiator is not running.

MQSVC_STATUS_STARTING
The channel initiator is in the process of initializing.

MQSVC_STATUS_RUNNING
The channel initiator is fully initialized and is running.

MQSVC_STATUS_STOPPING
The channel initiator is stopping.

CommandServerStatus **(MQCFIN)**
Status of the command server (parameter identifier: MQIACF_CMD_SERVER_STATUS).

The value can be:

MQSVC_STATUS_STARTING
The command server is in the process of initializing.

MQSVC_STATUS_RUNNING
The command server is fully initialized and is running.

MQSVC_STATUS_STOPPING
The command server is stopping.

ConnectionCount **(MQCFIN)**
Connection count (parameter identifier: MQIACF_CONNECTION_COUNT).

The current number of connections to the queue manager.

CurrentLog **(MQCFST)**
Log extent name (parameter identifier: MQCACF_CURRENT_LOG_EXTENT_NAME).

The name of the log extent that was being written to at the time of the Inquire command. If the queue manager is using circular logging, this parameter is blank.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

InstallationDesc **(MQCFST)**
Installation Description (parameter identifier: MQCA_INSTALLATION_DESC)

The installation description for this queue manager. Not valid on IBM i.

InstallationName **(MQCFST)**
Installation Name (parameter identifier: MQCA_INSTALLATION_NAME)

The installation name for this queue manager. Not valid on IBM i.

InstallationPath **(MQCFST)**
Installation Path (parameter identifier: MQCA_INSTALLATION_PATH)

The installation path for this queue manager. Not valid on IBM i.

LogPath **(MQCFST)**
Location of the recovery log extents (parameter identifier: MQCACF_LOG_PATH).

This parameter identifies the directory where log files are created by the queue manager.

The maximum length of the string is MQ_LOG_PATH_LENGTH.

MediaRecoveryLog **(MQCFST)**
Name of the oldest log extent required by the queue manager to perform media recovery (parameter identifier: MQCACF_MEDIA_LOG_EXTENT_NAME). This parameter is available only on queue managers using linear logging. If the queue manager is using circular logging, this parameter is blank.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

QMgrName **(MQCFST)**
Name of the local queue manager (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QMgrStatus **(MQCFIN)**
Current execution status of the queue manager (parameter identifier: MQIACF_Q_MGR_STATUS).

The value can be:

MQQMSTA_STARTING
The queue manager is initializing.

MQQMSTA_RUNNING
The queue manager is fully initialized and is running.

MQQMSTA_QUIESCING
The queue manager is quiescing.

RestartRecoveryLog **(MQCFST)**
Name of the oldest log extent required by the queue manager to perform restart recovery (parameter identifier: MQCACF_RESTART_LOG_EXTENT_NAME).

This parameter is available only on queue managers using linear logging. If the queue manager is using circular logging, this parameter is blank.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

StartDate **(MQCFST)**
Date when this queue manager was started (in the form `yyyy-mm-dd`) (parameter identifier: MQCACF_Q_MGR_START_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

StartTime **(MQCFST)**
Time when this queue manager was started (in the form hh:mm:ss) (parameter identifier: MQCACF_Q_MGR_START_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

Inquire Queue Names:

The Inquire Queue Names (MQCMD_INQUIRE_Q_NAMES) command inquires a list of queue names that match the generic queue name, and the optional queue type specified.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_Q_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED. MQQSGD_SHARED is permitted only in a shared queue environment.

QType **(MQCFIN)**
Queue type (parameter identifier: MQIA_Q_TYPE).

If present, this parameter limits the queue names returned to queues of the specified type. If this parameter is not present, queues of all types are eligible. The value can be:

MQQT_ALL
All queue types.

MQQT_LOCAL
Local queue.

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

The default value if this parameter is not specified is MQQT_ALL.

Inquire Queue Names (Response):

The response to the Inquire Queue Names (MQCMD_INQUIRE_Q_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified queue name. The response header is followed by the *QTypes* structure, with the same number of entries as the *QNames* structure. Each entry gives the type of the queue with the corresponding entry in the *QNames* structure.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Additionally, on z/OS only, the *QSGDispositions* parameter structure (with the same number of entries as the *QNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *QNames* structure.

Always returned:
QNames, *QSGDispositions*, *QTypes*

Returned if requested:
None

Response data

QNames **(MQCFSL)**
List of queue names (parameter identifier: MQCACF_Q_NAMES).

QSGDispositions **(MQCFIL)**
List of QSG dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter is valid on z/OS only. Possible values for fields in this structure are:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

QTypes **(MQCFIL)**
List of queue types (parameter identifier: MQIACF_Q_TYPES). Possible values for fields in this structure are:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

Inquire Queue Status:

The Inquire Queue Status (MQCMD_INQUIRE_Q_STATUS) command inquires about the status of a local WebSphere MQ queue. You must specify the name of a local queue for which you want to receive status information.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all queues having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters (Inquire Queue Status)

ByteStringFilterCommand **(MQCFBF)**
Byte string filter command descriptor. The parameter identifier must be MQBACF_EXTERNAL_UOW_ID or MQBACF_Q_MGR_UOW_ID. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFBF - PCF byte string filter parameter" on page 1219 for information about using this filter condition.

If you specify a byte string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter, or a string filter using the *StringFilterCommand* parameter.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is initiated when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is initiated on the queue manager on which it was entered.
- Queue manager name. The command is initiated on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be initiated.
- An asterisk (*). The command is initiated on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *QStatusAttrs* except MQIACF_ALL, MQIACF_MONITORING, and MQIACF_Q_TIME_INDICATOR. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a byte string filter using the *ByteStringFilterCommand* parameter or a string filter using the *StringFilterCommand* parameter.

OpenType **(MQCFIN)**
Queue status open type (parameter identifier: MQIACF_OPEN_TYPE).

It is always returned, regardless of the queue instance attributes requested.

The value can be:

MQQSOT_ALL
Selects status for queues that are open with any type of access.

MQQSOT_INPUT
Selects status for queues that are open for input.

MQQSOT_OUTPUT
Selects status for queues that are open for output.

The default value if this parameter if not specified is MQQSOT_ALL.

Filtering is not supported for this parameter.

QSGDispositon **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid only on z/OS. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

You cannot use *QSGDispositon* as a parameter to filter on.

QStatusAttrs **(MQCFIL)**
Queue status attributes (parameter identifier: MQIACF_Q_STATUS_ATTRS).

The attribute list can specify the following value on its own - default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

Where *StatusType* is MQIACF_Q_STATUS:

MQCA_Q_NAME
Queue name.

MQCACF_LAST_GET_DATE
Date of the last message successfully destructively read from the queue.

MQCACF_LAST_GET_TIME
Time of the last message successfully destructively read from the queue.

MQCACF_LAST_PUT_DATE
Date of the last message successfully put to the queue.

MQCACF_LAST_PUT_TIME
Time of the last message successfully put to the queue.

MQCACF_MEDIA_LOG_EXTENT_NAME
Identity of the oldest log extent required to perform media recovery of the queue.

On IBM i, this parameter identifies the name of the oldest journal receiver require to perform media recovery of the queue.

MQIA_CURRENT_Q_DEPTH
The current number of messages on the queue.

MQIA_MONITORING_Q
Current level of monitoring data collection.

MQIA_OPEN_INPUT_COUNT
The number of handles that are currently open for input for the queue.
MQIA_OPEN_INPUT_COUNT does not include handles that are open for browse.

MQIA_OPEN_OUTPUT_COUNT
The number of handles that are currently open for output for the queue.

MQIACF_HANDLE_STATE
Whether an API call is in progress.

MQIACF_MONITORING
All the queue status monitoring attributes. These attributes are:

- MQCACF_LAST_GET_DATE
- MQCACF_LAST_GET_TIME
- MQCACF_LAST_PUT_DATE
- MQCACF_LAST_PUT_TIME
- MQIA_MONITORING_Q
- MQIACF_OLDEST_MSG_AGE
- MQIACF_Q_TIME_INDICATOR

Filtering is not supported for this parameter.

MQIACF_OLDEST_MSG_AGE
Age of oldest message on the queue.

MQIACF_Q_TIME_INDICATOR
Indicator of the time that messages remain on the queue.

MQIACF_UNCOMMITTED_MSGS
The number of uncommitted messages on the queue.

Where *StatusType* is MQIACF_Q_HANDLE:

MQBACF_EXTERNAL_UOW_ID
Unit of recovery identifier assigned by the queue manager.

MQBACF_Q_MGR_UOW_ID
External unit of recovery identifier associated with the connection.

MQCA_Q_NAME
Queue name.

MQCACF_APPL_TAG
This parameter is a string containing the tag of the application connected to the queue manager.

MQCACF_ASID
Address-space identifier of the application identified by *ApplTag*. This parameter is valid on z/OS only.

MQCACF_PSB_NAME
Name of the program specification block (PSB) associated with the running IMS transaction. This parameter is valid on z/OS only.

MQCACF_PSTID
Identifier of the IMS program specification table (PST) for the connected IMS region. This parameter is valid on z/OS only.

MQCACF_TASK_NUMBER
CICS task number. This parameter is valid on z/OS only.

MQCACF_TRANSACTION_ID
CICS transaction identifier. This parameter is valid on z/OS only.

MQCACF_USER_IDENTIFIER
The user name of the application that has opened the specified queue.

MQCACH_CHANNEL_NAME
The name of the channel that has the queue open, if any.

MQCACH_CONNECTION_NAME
The connection name of the channel that has the queue open, if any.

MQIA_APPL_TYPE
The type of application that has the queue open.

MQIACF_OPEN_BROWSE
Open browse.

Filtering is not supported for this parameter.

MQIACF_OPEN_INPUT_TYPE
Open input type.

Filtering is not supported for this parameter.

MQIACF_OPEN_INQUIRE
Open inquire.

Filtering is not supported for this parameter.

MQIACF_OPEN_OPTIONS
The options used to open the queue.

If this parameter is requested, the following parameter structures are also returned:
- *OpenBrowse*
- *OpenInputType*
- *OpenInquire*
- *OpenOutput*
- *OpenSet*

Filtering is not supported for this parameter.

MQIACF_OPEN_OUTPUT
Open output.

Filtering is not supported for this parameter.

MQIACF_OPEN_SET
Open set.

Filtering is not supported for this parameter.

MQIACF_PROCESS_ID
The process identifier of the application that has opened the specified queue.

MQIACF_ASYNC_STATE

MQIACF_THREAD_ID
The thread identifier of the application that has opened the specified queue.

MQIACF_UOW_TYPE
Type of external unit of recovery identifier as seen by the queue manager.

StatusType **(MQCFIN)**
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE).

Specifies the type of status information required.

The value can be:

MQIACF_Q_STATUS
Selects status information relating to queues.

MQIACF_Q_HANDLE
Selects status information relating to the handles that are accessing the queues.

The default value, if this parameter is not specified, is MQIACF_Q_STATUS.

You cannot use *StatusType* as a parameter to filter on.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *QStatusAttrs* except MQCA_Q_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify a byte string filter using the *ByteStringFilterCommand* parameter or an integer filter using the *IntegerFilterCommand* parameter.

Error codes

This command might return the following error code in the response format header "Error codes applicable to all commands" on page 802 along with any additional pertinent values.

Reason **(MQLONG)**
The value can be:

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

Inquire Queue Status (Response):

The response to the Inquire Queue Status (MQCMD_INQUIRE_Q_STATUS) command consists of the response header followed by the *QName* structure and a set of attribute parameter structures determined by the value of *StatusType* in the Inquire command.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned:
QName, *QSGDisposition*, *StatusType*

Possible values of *StatusType* are:

MQIACF_Q_STATUS
Returns status information relating to queues.

MQIACF_Q_HANDLE
Returns status information relating to the handles that are accessing the queues.

Returned if requested and *StatusType* is MQIACF_Q_STATUS:
CurrentQDepth, *LastGetDate*, *LastGetTime*, *LastPutDate*, *LastPutTime*, *MediaRecoveryLogExtent*, *OldestMsgAge*, *OnQTime*, *OpenInputCount*, *OpenOutputCount*, *QueueMonitoring*, *UncommittedMsgs*

Returned if requested and *StatusType* is MQIACF_Q_HANDLE:
ApplDesc, *ApplTag*, *ApplType*, *ASId*, *AsynchronousState*, *ChannelName*, *ConnectionName*, *ExternalUOWId*, *HandleState*, *OpenOptions*, *ProcessId*, *PSBName*, *PSTId*, *QMgrUOWId*, *TaskNumber*, *ThreadId*, *TransactionId*, *UOWIdentifier*, *UOWType*, *UserIdentifier*

Response data if StatusType is MQIACF_Q_STATUS

CurrentQDepth **(MQCFIN)**
Current queue depth (parameter identifier: MQIA_CURRENT_Q_DEPTH).

LastGetDate **(MQCFST)**
Date on which the last message was destructively read from the queue (parameter identifier: MQCACF_LAST_GET_DATE).

The date, in the form `yyyy-mm-dd`, on which the last message was successfully read from the queue. The date is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_DATE_LENGTH.

LastGetTime **(MQCFST)**
Time at which the last message was destructively read from the queue (parameter identifier: MQCACF_LAST_GET_TIME).

The time, in the form `hh.mm.ss`, at which the last message was successfully read from the queue. The time is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_TIME_LENGTH.

LastPutDate **(MQCFST)**
Date on which the last message was successfully put to the queue (parameter identifier: MQCACF_LAST_PUT_DATE).

The date, in the form `yyyy-mm-dd`, on which the last message was successfully put to the queue. The date is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_DATE_LENGTH.

LastPutTime **(MQCFST)**
Time at which the last message was successfully put to the queue (parameter identifier: MQCACF_LAST_PUT_TIME).

The time, in the form `hh.mm.ss`, at which the last message was successfully put to the queue. The time is returned in the time zone in which the queue manager is running.

The maximum length of the string is MQ_TIME_LENGTH.

MediaRecoveryLogExtent **(MQCFST)**
Name of the oldest log extent required to perform media recovery of the queue (parameter identifier: MQCACF_MEDIA_LOG_EXTENT_NAME).

On IBM i, this parameter identifies the name of the oldest journal receiver required to perform media recovery of the queue.

The name returned is of the form `Snnnnnnn.LOG` and is not a fully qualified path name. The use of this parameter provides the ability for the name to be easily correlated with the messages issued, following an**rcdmqimg** command to identify those queues causing the media recovery LSN not to move forwards.

This parameter is valid on AIX, HP-UX, Linux, IBM i, Solaris, and Windows.

The maximum length of the string is MQ_LOG_EXTENT_NAME_LENGTH.

OldestMsgAge **(MQCFIN)**
Age of the oldest message (parameter identifier: MQIACF_OLDEST_MSG_AGE). Age, in seconds, of the oldest message on the queue.

If the value is unavailable, MQMON_NOT_AVAILABLE is returned. If the queue is empty, `0` is returned. If the value exceeds 999 999 999, it is returned as 999 999 999.

OnQTime **(MQCFIL)**
Indicator of the time that messages remain on the queue (parameter identifier: MQIACF_Q_TIME_INDICATOR). Amount of time, in microseconds, that a message spent on the queue. Two values are returned:

- A value based on recent activity over a short period.
- A value based on activity over a longer period.

Where no measurement is available, the value MQMON_NOT_AVAILABLE is returned. If the value exceeds 999 999 999, it is returned as 999 999 999.

OpenInputCount **(MQCFIN)**
Open input count (parameter identifier: MQIA_OPEN_INPUT_COUNT).

OpenOutputCount **(MQCFIN)**
Open output count (parameter identifier: MQIA_OPEN_OUTPUT_COUNT).

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Returns the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

QueueMonitoring **(MQCFIN)**
Current level of monitoring data collection for the queue (parameter identifier: MQIA_MONITORING_Q). The value can be:

MQMON_OFF
Monitoring for the queue is switched off.

MQMON_LOW
Low rate of data collection.

MQMON_MEDIUM
Medium rate of data collection.

MQMON_HIGH
High rate of data collection.

StatusType **(MQCFST)**
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE).

Specifies the type of status information.

UncommittedMsgs **(MQCFIN)**
The number of uncommitted changes (puts and gets) pending for the queue (parameter identifier: MQIACF_UNCOMMITTED_MSGS). The value can be:

MQQSUM_YES
On z/OS, there are one or more uncommitted changes pending.

MQQSUM_NO
There are no uncommitted changes pending.

n On platforms other than z/OS, an integer value indicating how many uncommitted changes are pending.

Response data if StatusType is MQIACF_Q_HANDLE

ApplDesc **(MQCFST)**
Application description (parameter identifier: MQCACF_APPL_DESC).

The maximum length is MQ_APPL_DESC_LENGTH.

ApplTag **(MQCFST)**
Open application tag (parameter identifier: MQCACF_APPL_TAG).

The maximum length of the string is MQ_APPL_TAG_LENGTH.

ApplType **(MQCFIN)**
Open application type (parameter identifier: MQIA_APPL_TYPE).

The value can be:

MQAT_QMGR
A queue manager process.

MQAT_CHANNEL_INITIATOR
The channel initiator.

MQAT_USER
A user application.

MQAT_BATCH
Application using a batch connection. MQAT_BATCH applies only to z/OS.

MQAT_RRS_BATCH
RRS-coordinated application using a batch connection. MQAT_RRS_BATCH applies only to z/OS.

MQAT_CICS
A CICS transaction. MQAT_CICS applies only to z/OS.

MQAT_IMS
An IMS transaction. MQAT_IMS applies only to z/OS.

MQAT_SYSTEM_EXTENSION
Application performing an extension of function that is provided by the queue manager.

ASId **(MQCFST)**
Address-space identifier (parameter identifier: MQCACF_ASID).

The 4-character address-space identifier of the application identified by *ApplTag*. It distinguishes duplicate values of *ApplTag*. This parameter applies only to z/OS.

The length of the string is MQ_ASID_LENGTH.

AsynchronousState **(MQCFIN)**
The state of the asynchronous consumer on this queue (parameter identifier: MQIACF_ASYNC_STATE).

The value can be:

MQAS_ACTIVE
An MQCB call has set up a function to call back to process messages asynchronously and the connection handle has been started so that asynchronous message consumption can proceed.

MQAS_INACTIVE
An MQCB call has set up a function to call back to process messages asynchronously but the connection handle has not yet been started, or has been stopped or suspended, so that asynchronous message consumption cannot currently proceed.

MQAS_SUSPENDED
The asynchronous consumption callback has been suspended so that asynchronous message consumption cannot currently proceed on this handle. This situation can be either because an MQCB or MQCTL call with *Operation* MQOP_SUSPEND has been issued against this object handle by the application, or because it has been suspended by the system. If it has been suspended by the system, as part of the process of suspending asynchronous message consumption the callback function is called with the reason code that describes the problem resulting in suspension. This situation is reported in the *Reason* field in the MQCBC structure passed to the callback. In order for asynchronous message consumption to proceed, the application must issue an MQCB or MQCTL call with *Operation* MQOP_RESUME.

MQAS_SUSPENDED_TEMPORARY
The asynchronous consumption callback has been temporarily suspended by the system so that asynchronous message consumption cannot currently proceed on this object handle. As part of the process of suspending asynchronous message consumption the callback function is called with the reason code that describes the problem resulting in suspension. This situation is reported in the *Reason* field in the MQCBC structure passed to the callback. The callback function is called again when asynchronous message consumption is resumed by the system after the temporary condition has been resolved.

MQAS_NONE
An MQCB call has not been issued against this handle, so no asynchronous message consumption is configured on this handle.

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Conname **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

ExternalUOWId **(MQCFBS)**
RRS unit-of-recovery identifier (parameter identifier: MQBACF_EXTERNAL_UOW_ID).

The RRS unit-of-recovery identifier associated with the handle. This parameter is valid only on z/OS only.

The length of the string is MQ_EXTERNAL_UOW_ID_LENGTH.

HandleState **(MQCFIN)**
State of the handle (parameter identifier: MQIACF_HANDLE_STATE).

The value can be:

MQHSTATE_ACTIVE
An API call from a connection is currently in progress for this object. For a queue, this condition can arise when an MQGET WAIT call is in progress.

If there is an MQGET SIGNAL outstanding, it does not mean, by itself, that the handle is active.

MQHSTATE_INACTIVE
No API call from a connection is currently in progress for this object. For a queue, this condition can arise when no MQGET WAIT call is in progress.

OpenBrowse **(MQCFIN)**
Open browse (parameter identifier: MQIACF_OPEN_BROWSE).

The value can be:

MQQSO_YES
The queue is open for browsing.

MQQSO_NO
The queue is not open for browsing.

OpenInputType **(MQCFIN)**
Open input type (parameter identifier: MQIACF_OPEN_INPUT_TYPE).

The value can be:

MQQSO_NO
The queue is not open for inputting.

MQQSO_SHARED
The queue is open for shared input.

MQQSO_EXCLUSIVE
The queue is open for exclusive input.

OpenInquire **(MQCFIN)**
Open inquire (parameter identifier: MQIACF_OPEN_INQUIRE).

The value can be:

MQQSO_YES
The queue is open for inquiring.

MQQSO_NO
The queue is not open for inquiring.

OpenOptions **(MQCFIN)**
Open options currently in force for the queue (parameter identifier: MQIACF_OPEN_OPTIONS).

OpenOutput **(MQCFIN)**
Open output (parameter identifier: MQIACF_OPEN_OUTPUT).

The value can be:

MQQSO_YES
The queue is open for output.

MQQSO_NO
The queue is not open for output.

OpenSet **(MQCFIN)**
Open set (parameter identifier: MQIACF_OPEN_SET).

The value can be:

MQQSO_YES
The queue is open for setting.

MQQSO_NO
The queue is not open for setting.

ProcessId **(MQCFIN)**
Open application process ID (parameter identifier: MQIACF_PROCESS_ID).

PSBName **(MQCFST)**
Program specification block (PSB) name (parameter identifier: MQCACF_PSB_NAME).

The 8-character name of the PSB associated with the running IMS transaction. This parameter is valid on z/OS only.

The length of the string is MQ_PSB_NAME_LENGTH.

PSTId **(MQCFST)**
Program specification table (PST) identifier (parameter identifier: MQCACF_PST_ID).

The 4-character identifier of the PST region identifier for the connected IMS region. This parameter is valid on z/OS only.

The length of the string is MQ_PST_ID_LENGTH.

QMgrUOWId **(MQCFBS)**
The unit of recovery assigned by the queue manager (parameter identifier: MQBACF_Q_MGR_UOW_ID).

On z/OS, this parameter is a 6-byte log RBA, displayed as 12 hexadecimal characters. On platforms other than z/OS, this parameter is an 8-byte transaction identifier, displayed as 16 hexadecimal characters.

The maximum length of the string is MQ_UOW_ID_LENGTH.

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Returns the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

StatusType **(MQCFST)**
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE).

Specifies the type of status information.

TaskNumber **(MQCFST)**
CICS task number (parameter identifier: MQCACF_TASK_NUMBER).

A 7-digit CICS task number. This parameter is valid on z/OS only.

The length of the string is MQ_TASK_NUMBER_LENGTH.

ThreadId **(MQCFIN)**
The thread ID of the open application (parameter identifier: MQIACF_THREAD_ID).

A value of zero indicates that the handle was opened by a shared connection. A handle created by a shared connection is logically open to all threads.

TransactionId **(MQCFST)**
CICS transaction identifier (parameter identifier: MQCACF_TRANSACTION_ID).

A 4-character CICS transaction identifier. This parameter is valid on z/OS only.

The length of the string is MQ_TRANSACTION_ID_LENGTH.

UOWIdentifier **(MQCFBS)**
The external unit of recovery associated with the connection (parameter identifier: MQBACF_EXTERNAL_UOW_ID).

This parameter is the recovery identifier for the unit of recovery. Its format is determined by the value of *UOWType*.

The maximum length of the string is MQ_UOW_ID_LENGTH.

UOWType **(MQCFIN)**
Type of external unit of recovery identifier as perceived by the queue manager (parameter identifier: MQIACF_UOW_TYPE).

The value can be:

MQUOWT_Q_MGR

MQUOWT_CICS
Valid only on z/OS.

MQUOWT_RRS
Valid only on z/OS.

MQUOWT_IMS
Valid only on z/OS.

MQUOWT_XA

UOWType identifies the *UOWIdentifier* type and not the type of the transaction coordinator. When the value of *UOWType* is MQUOWT_Q_MGR, the associated identifier is in *QMgrUOWId* (and not *UOWIdentifier*).

UserIdentifier **(MQCFST)**
Open application user name (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_MAX_USER_ID_LENGTH.

Inquire Service:

The Inquire Service (MQCMD_INQUIRE_SERVICE) command inquires about the attributes of existing WebSphere MQ services.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ServiceName **(MQCFST)**
Service name (parameter identifier: MQCA_SERVICE_NAME).

This parameter is the name of the service whose attributes are required. Generic service names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all services having names that start with the selected character string. An asterisk on its own matches all possible names.

The service name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ServiceAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

ServiceAttrs **(MQCFIL)**
Service attributes (parameter identifier: MQIACF_SERVICE_ATTRS).

The attribute list might specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
Date on which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCA_SERVICE_DESC
Description of service definition.

MQCA_SERVICE_NAME
Name of service definition.

MQCA_SERVICE_START_ARGS
Arguments to be passed to the service program.

MQCA_SERVICE_START_COMMAND
Name of program to run to start the service.

MQCA_SERVICE_STOP_ARGS
Arguments to be passed to the stop program to stop the service.

MQCA_STDERR_DESTINATION
Destination of standard error for the process.

MQCA_STDOUT_DESTINATION
Destination of standard output for the process.

MQCA_SERVICE_START_ARGS
Arguments to be passed to the service program.

MQIA_SERVICE_CONTROL
When the queue manager must start the service.

MQIA_SERVICE_TYPE
Mode in which the service is to run.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ServiceAttrs* except MQCA_SERVICE_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Inquire Service (Response):

The response to the Inquire Service (MQCMD_INQUIRE_SERVICE) command consists of the response header followed by the *ServiceName* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

If a generic service name was specified, one such message is generated for each service found.

Always returned:
ServiceName

Returned if requested:
AlterationDate, *AlterationTime*, *Arguments*, *ServiceDesc*, *ServiceType*, *StartArguments*, *StartCommand*, *StartMode*, *StderrDestination*, *StdoutDestination*, *StopArguments*, *StopCommand*

Response data

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date on which the information was last altered in the form `yyyy-mm-dd`.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time at which the information was last altered in the form `hh.mm.ss`.

ServiceDesc **(MQCFST)**
Description of service definition (parameter identifier: MQCA_SERVICE_DESC).

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceName **(MQCFST)**
Name of service definition (parameter identifier: MQCA_SERVICE_ NAME).

The maximum length of the string is MQ_SERVICE_NAME_LENGTH.

ServiceType **(MQCFIN)**
The mode in which the service is to run (parameter identifier: MQIA_SERVICE_TYPE).

The value can be:

MQSVC_TYPE_SERVER
Only one instance of the service can be executed at a time, with the status of the service made available by the Inquire Service Status command.

MQSVC_TYPE_COMMAND
Multiple instances of the service can be started.

StartArguments **(MQCFST)**
The arguments to be passed to the user program at queue manager startup (parameter identifier: MQCA_SERVICE_START_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StartCommand **(MQCFST)**
Service program name (parameter identifier: MQCA_SERVICE_START_COMMAND).

The name of the program which is to run.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

StartMode **(MQCFIN)**
Service mode (parameter identifier: MQIA_SERVICE_CONTROL).

Specifies how the service is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command.

MQSVC_CONTROL_Q_MGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The service is to be started at the same time as the queue manager is started, but is not requested to stop when the queue manager is stopped.

StderrDestination **(MQCFST)**
The path to a file to which the standard error (stderr) of the service program is to be redirected (parameter identifier: MQCA_STDERR_DESTINATION).

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StdoutDestination **(MQCFST)**
The path to a file to which the standard output (stdout) of the service program is to be redirected (parameter identifier: MQCA_STDOUT_DESTINATION).

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StopArguments **(MQCFST)**
The arguments to be passed to the stop program when instructed to stop the service (parameter identifier: MQCA_SERVICE_STOP_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StopCommand **(MQCFST)**
Service program stop command (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

This parameter is the name of the program that is to run when the service is requested to stop.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

Inquire Service Status:

The Inquire Service Status (MQCMD_INQUIRE_SERVICE_STATUS) command inquires about the status of one or more WebSphere MQ service instances.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ServiceName **(MQCFST)**
Service name (parameter identifier: MQCA_SERVICE_NAME).

Generic service names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all services having names that start with the selected character string. An asterisk on its own matches all possible names.

The service name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Optional parameters (Inquire Service Status)

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *ServiceStatusAttrs* except MQIACF_ALL. Use this parameter to restrict the output from the command by specifying a filter condition. See “MQCFIF - PCF integer filter parameter” on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

ServiceStatusAttrs **(MQCFIL)**
Service status attributes (parameter identifier: MQIACF_SERVICE_STATUS_ATTRS).

The attribute list can specify the following value on its own - is the default value used if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_SERVICE_DESC
Description of service definition.

MQCA_SERVICE_NAME
Name of service definition.

MQCA_SERVICE_START_ARGS
The arguments to pass to the service program.

MQCA_SERVICE_START_COMMAND
The name of the program to run to start the service.

MQCA_SERVICE_STOP_ARGS
The arguments to pass to the stop command to stop the service.

MQCA_SERVICE_STOP_COMMAND
The name of the program to run to stop the service.

MQCA_STDERR_DESTINATION
Destination of standard error for the process.

MQCA_STDOUT_DESTINATION
Destination of standard output for the process.

MQCACF_SERVICE_START_DATE
The date on which the service was started.

MQCACF_SERVICE_START_TIME
The time at which the service was started.

MQIA_SERVICE_CONTROL
How the service is to be started and stopped.

MQIA_SERVICE_TYPE
The mode in which the service is to run.

MQIACF_PROCESS_ID
The process identifier of the operating system task under which this service is executing.

MQIACF_SERVICE_STATUS
Status of the service.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *ServiceStatusAttrs* except MQCA_SERVICE_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_SERV_STATUS_NOT_FOUND
Service status not found.

Inquire Service Status (Response):

The response to the Inquire Service Status (MQCMD_INQUIRE_SERVICE_STATUS) command consists of the response header followed by the *ServiceName* structure and the requested combination of attribute parameter structures.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

If a generic service name was specified, one such message is generated for each service found.

Always returned:
ServiceName

Returned if requested:
ProcessId, *ServiceDesc*, *StartArguments*, *StartCommand*, *StartDate*, *StartMode*, *StartTime*, *Status*, *StderrDestination*, *StdoutDestination*, *StopArguments*, *StopCommand*

Response data

ProcessId **(MQCFIN)**
Process identifier (parameter identifier: MQIACF_PROCESS_ID).

The operating system process identifier associated with the service.

ServiceDesc **(MQCFST)**
Description of service definition (parameter identifier: MQCACH_SERVICE_DESC).

The maximum length of the string is MQ_SERVICE_DESC_LENGTH.

ServiceName **(MQCFST)**
Name of the service definition (parameter identifier: MQCA_SERVICE_NAME).

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

StartArguments **(MQCFST)**
Arguments to be passed to the program on startup (parameter identifier: MQCA_SERVICE_START_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StartCommand **(MQCFST)**
Service program name (parameter identifier: MQCA_SERVICE_START_COMMAND).

Specifies the name of the program which is to run.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

StartDate **(MQCFST)**
Start date (parameter identifier: MQIACF_SERVICE_START_DATE).

The date, in the form `yyyy-mm-dd`, on which the service was started.

The maximum length of the string is MQ_DATE_LENGTH

StartMode **(MQCFIN)**
Service mode (parameter identifier: MQIACH_SERVICE_CONTROL).

How the service is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The service is not to be started automatically or stopped automatically. It is to be controlled by user command.

MQSVC_CONTROL_Q_MGR
The service is to be started and stopped at the same time as the queue manager is started and stopped.

MQSVC_CONTROL_Q_MGR_START
The service is to be started at the same time as the queue manager is started, but is not request to stop when the queue manager is stopped.

StartTime **(MQCFST)**
Start date (parameter identifier: MQIACF_SERVICE_START_TIME).

The time, in the form `hh.mm.ss`, at which the service was started.

The maximum length of the string is MQ_TIME_LENGTH

Status **(MQCFIN)**
Service status (parameter identifier: MQIACF_SERVICE_STATUS).

The status of the service. The value can be:

MQSVC_STATUS_STARTING
The service is in the process of initializing.

MQSVC_STATUS_RUNNING
The service is running.

MQSVC_STATUS_STOPPING
The service is stopping.

StderrDestination **(MQCFST)**
Specifies the path to a file to which the standard error (stderr) of the service program is to be redirected (parameter identifier: MQCA_STDERR_DESTINATION).

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StdoutDestination **(MQCFST)**
Specifies the path to a file to which the standard output (stdout) of the service program is to be redirected (parameter identifier: MQCA_STDOUT_DESTINATION).

The maximum length of the string is MQ_SERVICE_PATH_LENGTH.

StopArguments **(MQCFST)**
Specifies the arguments to be passed to the stop program when instructed to stop the service (parameter identifier: MQCA_SERVICE_STOP_ARGS).

The maximum length of the string is MQ_SERVICE_ARGS_LENGTH.

StopCommand **(MQCFST)**
Service program stop command (parameter identifier: MQCA_SERVICE_STOP_COMMAND).

This parameter is the name of the program that is to run when the service is requested to stop.

The maximum length of the string is MQ_SERVICE_COMMAND_LENGTH.

Inquire Subscription:

The Inquire Subscription (MQCMD_INQUIRE_SUBSCRIPTION) command inquires about the attributes of a subscription.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

SubName **(MQCFST)**
The unique identifier of the application for a subscription (parameter identifier: MQCACF_SUB_NAME).

If *SubName* is not provided, *SubId* must be specified to identify the subscription to be inquired.

The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId **(MQCFBS)**
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

Specifies the unique internal subscription identifier. If the queue manager is generating the CorrelId for a subscription, then the *SubId* is used as the *DestinationCorrelId*.

You must supply a value for *SubId* if you have not supplied a value for *SubName*.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- A queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- An asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

Durable **(MQCFIN)**
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_YES
Information about durable subscriptions only is displayed.

MQSUB_DURABLE_NO
Information about nondurable subscriptions only is displayed.

MQSUB_DURABLE_ALL
Information about all subscriptions is displayed.

SubscriptionAttrs **(MQCFIL)**
Subscription attributes (parameter identifier: MQIACF_SUB_ATTRS).

Use one of the following parameters to select the attributes you want to display:

- ALL to display all attributes.
- SUMMARY to display a subset of the attributes (see MQIACF_SUMMARY for a list).
- Any of the following parameters individually or in combination.

MQIACF_ALL
All attributes.

MQIACF_SUMMARY
Use this parameter to display:

- MQBACF_DESTINATION_CORREL_ID
- MQBACF_SUB_ID
- MQCACF_DESTINATION
- MQCACF_DESTINATION_Q_MGR
- MQCACF_SUB_NAME
- MQCA_TOPIC_STRING
- MQIACF_SUB_TYPE

MQBACF_ACCOUNTING_TOKEN
The accounting token passed by the subscriber for propagation into messages sent to this subscription in the AccountingToken field of the MQMD.

MQBACF_DESTINATION_CORREL_ID
The CorrelId used for messages sent to this subscription.

MQBACF_SUB_ID
The internal unique key identifying a subscription.

MQCA_ALTERATION_DATE
The date of the most recent MQSUB with MQSO_ALTER or ALTER SUB command.

MQCA_ALTERATION_TIME
The time of the most recent MQSUB with MQSO_ALTER or ALTER SUB command.

MQCA_CREATION_DATE
The date of the first MQSUB command that caused this subscription to be created.

MQCA_CREATION_TIME
The time of the first MQSUB that caused this subscription to be created.

MQCA_TOPIC_STRING
The resolved topic string the subscription is for.

MQCACF_APPL_IDENTITY_DATA
The identity data passed by the subscriber for propagation into messages sent to this subscription in the ApplIdentity field of the MQMD.

MQCACF_DESTINATION
The destination for messages published to this subscription.

MQCACF_DESTINATION_Q_MGR
The destination queue manager for messages published to this subscription.

MQCACF_SUB_NAME
The unique identifier of an application for a subscription.

MQCACF_SUB_SELECTOR
The SQL 92 selector string to be applied to messages published on the named topic to select whether they are eligible for this subscription.

MQCACF_SUB_USER_DATA
The user data associated with the subscription.

MQCACF_SUB_USER_ID
The userid that owns the subscription. MQCACF_SUB_USER_ID is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription.

MQCA_TOPIC_NAME
The name of the topic object that identifies a position in the topic hierarchy to which the topic string is concatenated.

MQIACF_DESTINATION_CLASS
Indicated whether this subscription is a managed subscription.

MQIACF_DURABLE_SUBSCRIPTION
Whether the subscription is durable, persisting over queue manager restart.

MQIACF_EXPIRY
The time to live from creation date and time.

MQIACF_PUB_PRIORITY
The priority of the messages sent to this subscription.

MQIACF_PUBSUB_PROPERTIES
The manner in which publish/subscribe related message properties are added to messages sent to this subscription.

MQIACF_REQUEST_ONLY
Indicates whether the subscriber polls for updates by using MQSUBRQ API, or whether all publications are delivered to this subscription.

MQIACF_SUB_TYPE
The type of subscription - how it was created.

MQIACF_SUBSCRIPTION_SCOPE
Whether the subscription forwards messages to all other queue managers directly connected by using a Publish/Subscribe collective or hierarchy, or the subscription forwards messages on this topic within this queue manager only.

MQIACF_SUB_LEVEL
The level within the subscription interception hierarchy at which this subscription is made.

MQIACF_VARIABLE_USER_ID
Users other than the creator of this subscription that can connect to it (subject to topic and destination authority checks).

MQIACF_WILDCARD_SCHEMA
The schema to be used when interpreting wildcard characters in the topic string.

SubscriptionType **(MQCFIN)**
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_SUB_TYPE).

MQSUBTYPE_ADMIN
Subscriptions which have been created by an admin interface or modified by an admin interface are selected.

MQSUBTYPE_ALL
All subscription types are displayed.

MQSUBTYPE_API
Subscriptions created by applications by way of the WebSphere MQ API are displayed.

MQSUBTYPE_PROXY
System created subscriptions relating to inter-queue manager subscriptions are displayed.

MQSUBTYPE_USER
USER subscriptions (with SUBTYPE of either ADMIN or API) are displayed.
MQSUBTYPE_USER is the default value.

Inquire Subscription (Response):

The response to the Inquire Subscription (MQCMD_INQUIRE_SUBSCRIPTION) command consists of the response header followed by the *SubId* and *SubName* structures, and the requested combination of attribute parameter structures (where applicable).

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned
SubID, *SubName*

Returned if requested
AlterationDate, *AlterationTime*, *CreationDate*, *CreationTime*, *Destination*, *DestinationClass*, *DestinationCorrelId*, *DestinationQueueManager*, *Expiry*, *PublishedAccountingToken*, *PublishedApplicationIdentityData*, *PublishPriority*, *PublishSubscribeProperties*, *Requestonly*, *Selector*, *SelectorType*,*SubscriptionLevel*,*SubscriptionScope*, *SubscriptionType*, *SubscriptionUser*, *TopicObject*, *TopicString*, *Userdata*, *VariableUser*, *WildcardSchema*

Response Data

AlterationDate **(MQCFST)**
The date of the most recent **MQSUB** or **Change Subscription** command that modified the properties of the subscription (parameter identifier: MQCA_ALTERATION_DATE).

AlterationTime **(MQCFST)**
The time of the most recent **MQSUB** or **Change Subscription** command that modified the properties of the subscription (parameter identifier: MQCA_ALTERATION_TIME).

CreationDate **(MQCFST)**
The creation date of the subscription, in the form yyyy-mm-dd (parameter identifier: MQCA_CREATION_DATE).

CreationTime **(MQCFST)**
The creation time of the subscription, in the form hh.mm.ss (parameter identifier: MQCA_CREATION_TIME).

Destination **(MQCFST)**
Destination (parameter identifier: MQCACF_DESTINATION).

Specifies the name of the alias, local, remote, or cluster queue to which messages for this subscription are put.

DestinationClass **(MQCFIN)**
Destination class (parameter identifier: MQIACF_DESTINATION_CLASS).

Whether the destination is managed.

The value can be:

MQDC_MANAGED
The destination is managed.

MQDC_PROVIDED
The destination queue is as specified in the *Destination* field.

DestinationCorrelId **(MQCFBS)**
Destination correlation identifier (parameter identifier: MQBACF_DESTINATION_CORREL_ID).

A correlation identifier that is placed in the *CorrelId* field of the message descriptor for all the messages sent to this subscription.

The maximum length is MQ_CORREL_ID_LENGTH.

DestinationQueueManager **(MQCFST)**
Destination queue manager (parameter identifier: MQCACF_DESTINATION_Q_MGR).

Specifies the name of the destination queue manager, either local or remote, to which messages for the subscription are forwarded.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Durable **(MQCFIN)**
Whether this subscription is a durable subscription (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

The value can be:

MQSUB_DURABLE_YES
The subscription persists, even if the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. The queue manager reinstates the subscription during restart.

MQSUB_DURABLE_NO
The subscription is non-durable. The queue manager removes the subscription when the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. If the subscription has a destination class (DESTCLAS) of MANAGED, the queue manager removes any messages not yet consumed when it closes the subscription.

Expiry **(MQCFIN)**
The time, in tenths of a second, at which a subscription expires after its creation date and time (parameter identifier: MQIACF_EXPIRY).

A value of unlimited means that the subscription never expires.

After a subscription has expired it becomes eligible to be discarded by the queue manager and receives no further publications.

PublishedAccountingToken **(MQCFBS)**
Value of the accounting token used in the *AccountingToken* field of the message descriptor (parameter identifier: MQBACF_ACCOUNTING_TOKEN).

The maximum length of the string is MQ_ACCOUNTING_TOKEN_LENGTH.

PublishedApplicationIdentityData **(MQCFST)**
Value of the application identity data used in the *ApplIdentityData* field of the message descriptor (parameter identifier: MQCACF_APPL_IDENTITY_DATA.

The maximum length of the string is MQ_APPL_IDENTITY_DATA_LENGTH.

PublishPriority **(MQCFIN)**
The priority of messages sent to this subscription (parameter identifier: MQIACF_PUB_PRIORITY).

The value can be:

MQPRI_PRIORITY_AS_PUBLISHED
The priority of messages sent to this subscription is taken from that priority supplied to the published message. MQPRI_PRIORITY_AS_PUBLISHED is the supplied default value.

MQPRI_PRIORITY_AS_QDEF
The priority of messages sent to this subscription is determined by the default priority of the queue defined as a destination.

0-9 An integer value providing an explicit priority for messages sent to this subscription.

PublishSubscribeProperties **(MQCFIN)**
Specifies how publish/subscribe related message properties are added to messages sent to this subscription (parameter identifier: MQIACF_PUBSUB_PROPERTIES).

The value can be:

MQPSPROP_NONE
Publish/subscribe properties are not added to the messages. MQPSPROP_NONE is the supplied default value.

MQPSPROP_MSGPROP
Publish/subscribe properties are added as PCF attributes.

MQPSPROP_COMPAT
If the original publication is a PCF message, then the publish/subscribe properties are added as PCF attributes. Otherwise, publish/subscribe properties are added within an MQRFH version 1 header. This method is compatible with applications coded for use with previous versions of WebSphere MQ.

MQPSPROP_RFH2
Publish/subscribe properties are added within an MQRFH version 2 header. This method is compatible with applications coded for use with WebSphere Message Brokers.

Requestonly **(MQCFIN)**
Indicates whether the subscriber polls for updates using the MQSUBRQ API call, or whether all publications are delivered to this subscription (parameter identifier: MQIACF_REQUEST_ONLY).

The value can be:

MQRU_PUBLISH_ALL
All publications on the topic are delivered to this subscription.

MQRU_PUBLISH_ON_REQUEST
Publications are only delivered to this subscription in response to an MQSUBRQ API call.

Selector **(MQCFST)**
Specifies the selector applied to messages published to the topic (parameter identifier: MQCACF_SUB_SELECTOR).

Only those messages that satisfy the selection criteria are put to the destination specified by this subscription.

SelectorType **(MQCFIN)**
The type of selector string that has been specified (parameter identifier: MQIACF_SELECTOR_TYPE).

The value can be:

MQSELTYPE_NONE
No selector has been specified.

MQSELTYPE_STANDARD
The selector references only the properties of the message, not its content, using the standard WebSphere MQ selector syntax. Selectors of this type are to be handled internally by the queue manager.

MQSELTYPE_EXTENDED
The selector uses extended selector syntax, typically referencing the content of the message. Selectors of this type cannot be handled internally by the queue manager; extended selectors can be handled only by another program such as WebSphere Message Broker.

SubID **(MQCFBS)**
The internal, unique key identifying a subscription (parameter identifier: MQBACF_SUB_ID).

SubscriptionLevel **(MQCFIN)**
The level within the subscription interception hierarchy at which this subscription is made (parameter identifier: MQIACF_SUB_LEVEL).

The value can be:

0 - 9 An integer in the range 0-9. The default value is 1. Subscribers with a subscription level of 9 will intercept publications before they reach subscribers with lower subscription levels.

SubscriptionScope **(MQCFIN)**
Determines whether this subscription is passed to other queue managers in the network (parameter identifier: MQIACF_SUBSCRIPTION_SCOPE).

The value can be:

MQTSCOPE_ALL
The subscription is forwarded to all queue managers directly connected through a publish/subscribe collective or hierarchy. MQTSCOPE_ALL is the supplied default value.

MQTSCOPE_QMGR
The subscription only forwards messages published on the topic within this queue manager.

SubscriptionType **(MQCFIN)**
Indicates how the subscription was created (parameter identifier: MQIACF_SUB_TYPE).

MQSUBTYPE_PROXY
An internally created subscription used for routing publications through a queue manager.

MQSUBTYPE_ADMIN
Created using **DEF SUB** MQSC or PCF command. This **SUBTYPE** also indicates that a subscription has been modified using an administrative command.

MQSUBTYPE_API
Created using an **MQSUB** API request.

SubscriptionUser **(MQCFST)**
The userid that 'owns' this subscription. This parameter is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription. (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

TopicObject **(MQCFST)**
The name of a previously defined topic object from which is obtained the topic name for the subscription (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

TopicString **(MQCFST)**
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING).

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Userdata **(MQCFST)**
User data (parameter identifier: MQCACF_SUB_USER_DATA).

Specifies the user data associated with the subscription

The maximum length of the string is MQ_USER_DATA_LENGTH.

VariableUser **(MQCFIN)**
Specifies whether a user other than the one who created the subscription, that is, the user shown in *SubscriptionUser* can take over the ownership of the subscription (parameter identifier: MQIACF_VARIABLE_USER_ID).

The value can be:

MQVU_ANY_USER
Any user can take over the ownership. MQVU_ANY_USER is the supplied default value.

MQVU_FIXED_USER
No other user can take over the ownership.

WildcardSchema **(MQCFIN)**
Specifies the schema to be used when interpreting any wildcard characters contained in the *TopicString* (parameter identifier: MQIACF_WILDCARD_SCHEMA).

The value can be:

MQWS_CHAR
Wildcard characters represent portions of strings; it is for compatibility with WebSphere MQ V6.0 broker.

MQWS_TOPIC
Wildcard characters represent portions of the topic hierarchy; this is for compatibility with WebSphere Message Brokers. MQWS_TOPIC is the supplied default value.

Inquire Subscription Status:

The Inquire Subscription Status (MQCMD_INQUIRE_SUB_STATUS) command inquires about the status of a subscription.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

SubName **(MQCFST)**
The unique identifier of an application for a subscription (parameter identifier: MQCACF_SUB_NAME).

If *SubName* is not provided, *SubId* must be specified to identify the subscription to be inquired.

The maximum length of the string is MQ_SUB_NAME_LENGTH.

SubId **(MQCFBS)**
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

Specifies the unique internal subscription identifier. If the queue manager is generating the CorrelId for a subscription, then the *SubId* is used as the *DestinationCorrelId*.

You must supply a value for *SubId* if you have not supplied a value for *SubName*.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is processed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command is processed on the queue manager on which it was entered.
- A queue manager name. The command is processed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- An asterisk (*). The command is processed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter on which to filter.

Durable **(MQCFIN)**
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_YES
Information about durable subscriptions only is displayed. MQSUB_DURABLE_YES is the default.

MQSUB_DURABLE_NO
Information about non-durable subscriptions only is displayed.

SubscriptionType **(MQCFIN)**
Specify this attribute to restrict the type of subscriptions which are displayed (parameter identifier: MQIACF_SUB_TYPE).

MQSUBTYPE_ADMIN
Subscriptions which have been created by an admin interface or modified by an admin interface are selected.

MQSUBTYPE_ALL
All subscription types are displayed.

MQSUBTYPE_API
Subscriptions created by applications through a WebSphere MQ API call are displayed.

MQSUBTYPE_PROXY
System created subscriptions relating to inter-queue-manager subscriptions are displayed.

MQSUBTYPE_USER
USER subscriptions (with SUBTYPE of either ADMIN or API) are displayed. MQSUBTYPE_USER is the default value.

StatusAttrs **(MQCFIL)**
Subscription status attributes (parameter identifier: MQIACF_SUB_STATUS_ATTRS).

To select the attributes you want to display you can specify;

- ALL to display all attributes.
- any of the following parameters individually or in combination.

MQIACF_ALL
All attributes.

MQBACF_CONNECTION_ID
The currently active *ConnectionID* that has opened the subscription.

MQIACF_DURABLE_SUBSCRIPTION
Whether the subscription is durable, persisting over queue manager restart.

MQCACF_LAST_MSG_DATE
The date that a message was last sent to the destination specified by the subscription.

MQCACF_LAST_MSG_TIME
The time when a message was last sent to the destination specified by the subscription.

MQIACF_MESSAGE_COUNT
The number of messages put to the destination specified by the subscription.

MQCA_RESUME_DATE
The date of the most recent MQSUB command that connected to the subscription.

MQCA_RESUME_TIME
The time of the most recent MQSUB command that connected to the subscription.

MQIACF_SUB_TYPE
The type of subscription - how it was created.

MQCACF_SUB_USER_ID
The userid owns the subscription.

Inquire Subscription Status (Response):

The response to the Inquire Subscription Status (MQCMD_INQUIRE_SUB_STATUS) command consists of the response header followed by the *SubId* and *SubName* structures, and the requested combination of attribute parameter structures (where applicable).

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned
SubID, *SubName*

Returned if requested
ActiveConnection, *Durable*, *LastPublishDate*, *LastPublishTime*, *MCastRelIndicator*, *NumberMsgs*, *ResumeDate*, *ResumeTime*, *SubType*, *TopicString*

Response Data

ActiveConnection **(MQCFBS)**
The *ConnId* of the *HConn* that currently has this subscription open (parameter identifier: MQBACF_CONNECTION_ID).

Durable **(MQCFIN)**
A durable subscription is not deleted when the creating application closes its subscription handle (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_NO
The subscription is removed when the application that created it is closed or disconnected from the queue manager.

MQSUB_DURABLE_YES
The subscription persists even when the creating application is no longer running or has been disconnected. The subscription is reinstated when the queue manager restarts.

LastMessageDate **(MQCFST)**
The date that a message was last sent to the destination specified by the subscription (parameter identifier: MQCACF_LAST_MSG_DATE).

LastMessageTime **(MQCFST)**
The time when a message was last sent to the destination specified by the subscription (parameter identifier: MQCACF_LAST_MSG_TIME).

MCastRelIndicator **(MQCFIN)**
The multicast reliability indicator (parameter identifier: MQIACF_MCAST_REL_INDICATOR).

NumberMsgs **(MQCFIN)**
The number of messages put to the destination specified by this subscription (parameter identifier: MQIACF_MESSAGE_COUNT).

ResumeDate **(MQCFST)**
The date of the most recent **MQSUB** API call that connected to the subscription (parameter identifier: MQCA_RESUME_DATE).

ResumeTime **(MQCFST)**
The time of the most recent **MQSUB** API call that connected to the subscription (parameter identifier: MQCA_RESUME_TIME).

SubscriptionUser **(MQCFST)**
The userid that 'owns' this subscription. This parameter is either the userid associated with the creator of the subscription, or, if subscription takeover is permitted, the userid which last took over the subscription. (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

SubID **(MQCFBS)**
The internal, unique key identifying a subscription (parameter identifier: MQBACF_SUB_ID).

SubName **(MQCFST)**
The unique identifier of a subscription (parameter identifier: MQCACF_SUB_NAME).

SubType **(MQCFIN)**
Indicates how the subscription was created (parameter identifier: MQIA_SUB_TYPE).

MQSUBTYPE_PROXY
An internally created subscription used for routing publications through a queue manager.

MQSUBTYPE_ADMIN
Created using the **DEF SUB** MQSC or **Create Subscription**PCF command. This Subtype also indicates that a subscription has been modified using an administrative command.

MQSUBTYPE_API
Created using an **MQSUB** API call.

TopicString **(MQCFST)**
The resolved topic string (parameter identifier: MQCA_TOPIC_STRING). The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Inquire Topic:

The Inquire Topic (MQCMD_INQUIRE_TOPIC) command inquires about the attributes of existing IBM WebSphere MQ administrative topic objects

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

TopicName **(MQCFST)**
Administrative topic object name (parameter identifier: MQCA_TOPIC_NAME).

Specifies the name of the administrative topic object about which information is to be returned. Generic topic object names are supported. A generic name is a character string followed by an asterisk (*). For example, ABC* selects all administrative topic objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Optional parameters

ClusterInfo **(MQCFIN)**
Cluster information (parameter identifier: MQIACF_CLUSTER_INFO).

This parameter requests that, in addition to information about attributes of topics defined on this queue manager, cluster information about these topics and other topics in the repository that match the selection criteria is returned.

In this case, there might be multiple topics with the same name returned.

You can set this parameter to any integer value: the value used does not affect the response to the command.

The cluster information is obtained locally from the queue manager.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use *CommandScope* as a parameter to filter on.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor. The parameter identifier must be any integer type parameter allowed in *TopicAttrs* except MQIACF_ALL.

Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFIF - PCF integer filter parameter" on page 1224 for information about using this filter condition.

If you specify an integer filter, you cannot also specify a string filter using the *StringFilterCommand* parameter.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as either MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

You cannot use *QSGDisposition* as a parameter to filter on.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed in *TopicAttrs* except MQCA_TOPIC_NAME. Use this parameter to restrict the output from the command by specifying a filter condition. See "MQCFSF - PCF string filter parameter" on page 1231 for information about using this filter condition.

If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

TopicAttrs **(MQCFIL)**
Topic object attributes (parameter identifier: MQIACF_TOPIC_ATTRS).

The attribute list can specify the following value on its own - default value if the parameter is not specified:

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_ALTERATION_DATE
The date on which the information was last altered.

MQCA_ALTERATION_TIME
The time at which the information was last altered.

MQCA_CLUSTER_NAME
The cluster that is to be used for the propagation of publications and subscription to publish/subscribe cluster-connected queue managers for this topic.

MQCA_CLUSTER_DATE
The date on which this information became available to the local queue manager.

MQCA_CLUSTER_TIME
The time at which this information became available to the local queue manager.

MQCA_CLUSTER_Q_MGR_NAME
Queue manager that hosts the topic.

MQCA_CUSTOM
The custom attribute for new features.

MQCA_MODEL_DURABLE_Q
Name of the model queue for durable managed subscriptions.

MQCA_MODEL_NON_DURABLE_Q
Name of the model queue for non-durable managed subscriptions.

MQCA_TOPIC_DESC
Description of the topic object.

MQCA_TOPIC_NAME
Name of the topic object.

MQCA_TOPIC_STRING
The topic string for the topic object.

MQIA_DEF_PRIORITY
Default message priority.

MQIA_DEF_PUT_RESPONSE_TYPE
Default put response.

MQIA_DURABLE_SUB
Whether durable subscriptions are permitted.

MQIA_INHIBIT_PUB
Whether publications are allowed.

MQIA_INHIBIT_SUB
Whether subscriptions are allowed.

MQIA_NPM_DELIVERY
The delivery mechanism for non-persistent messages.

MQIA_PM_DELIVERY
The delivery mechanism for persistent messages.

MQIA_PROXY_SUB
Whether a proxy subscription is to be sent for this topic, even if no local subscriptions exist.

MQIA_PUB_SCOPE
Whether this queue manager propagates publications to queue managers as part of a hierarchy or a publish/subscribe cluster.

MQIA_SUB_SCOPE
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or a publish/subscribe cluster.

MQIA_TOPIC_DEF_PERSISTENCE
Default message persistence.

MQIA_USE_DEAD_LETTER_Q
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue.

TopicType **(MQCFIN)**
Cluster information (parameter identifier: MQIA_TOPIC_TYPE).

If this parameter is present, eligible queues are limited to the specified type. Any attribute selector that is specified in the TopicAttrs list and that is valid only for topics of different type is ignored; no error is raised.

If this parameter is not present (or if MQIACF_ALL is specified), queues of all types are eligible. Each attribute specified must be a valid topic attribute selector (that is, it must be in the following list), but it need not be applicable to all or any of the topics returned. Topic attribute selectors that are valid but not applicable to the queue are ignored; no error messages occur and no attribute is returned.

The value can be:

MQTOPT_ALL
All topic types are displayed. MQTOPT_ALL includes cluster topics, if ClusterInfo is also specified. MQTOPT_ALL is the default value.

MQTOPT_CLUSTER
Topics that are defined in publish/subscribe clusters are returned.

MQTOPT_LOCAL
Locally defined topics are displayed.

Inquire Topic (Response):

The response to the Inquire Topic (MQCMD_INQUIRE_TOPIC) command consists of the response header followed by the *TopicName* structure (and on z/OS only, the *QSG Disposition* structure), and the requested combination of attribute parameter structures (where applicable).

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned:
TopicName, *TopicType*, *QSGDisposition*

Returned if requested:
AlterationDate, *AlterationTime*, *ClusterName*, *Custom*, *DefPersistence*, *DefPriority*, *DefPutResponse*, *DurableModelQName*, *DurableSubscriptions*, *InhibitPublications*, *InhibitSubscriptions*, *NonDurableModelQName*, *NonPersistentMsgDelivery*, *PersistentMsgDelivery*, *ProxySubscriptions*, *PublicationScope*, *QMgrName*, *SubscriptionScope*, *TopicDesc*, *TopicString*, *UseDLQ*, *WildcardOperation*

Response data

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss`.

ClusterName **(MQCFST)**
The name of the cluster to which this topic belongs (parameter identifier: MQCA_CLUSTER_NAME).

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

The value can be as follows:

Blank This topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers.

Blank is the default value for this parameter if no value is specified.

String This topic belongs to the indicated cluster.

Additionally, if PublicationScope or SubscriptionScope is set to MQSCOPE_ALL, this cluster is to be used for the propagation of publications and subscriptions, for this topic, to publish/subcribe cluster-connected queue managers.

Custom **(MQCFST)**
Custom attribute for new features (parameter identifier: MQCA_CUSTOM).

This attribute is reserved for the configuration of new features before separate attributes have been introduced. It can contain the values of zero or more attributes as pairs of attribute name and value, separated by at least one space. The attribute name and value pairs have the form `NAME(VALUE)`.

This description will be updated when features using this attribute are introduced.

DefPersistence **(MQCFIN)**
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE).

The value can be:

MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority **(MQCFIN)**
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

DefPutResponse **(MQCFIN)**
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The value can be:

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName **(MQCFST)**
Name of the model queue to be used for durable managed subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions **(MQCFIN)**
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB).

The value can be:

MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.

MQSUB_DURABLE
Durable subscriptions are permitted.

MQSUB_NON_DURABLE
Durable subscriptions are not permitted.

InhibitPublications **(MQCFIN)**
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB).

The value can be:

MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_PUB_INHIBITED
Publications are inhibited for this topic.

MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions **(MQCFIN)**
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB).

The value can be:

MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.

MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

NonDurableModelQName **(MQCFST)**
Name of the model queue to be used for non-durable managed subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonPersistentMsgDelivery **(MQCFIN)**
The delivery mechanism for non-persistent messages published to this topic (parameter identifier: MQIA_NPM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

PersistentMsgDelivery **(MQCFIN)**
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ProxySubscriptions **(MQCFIN)**
Whether a proxy subscription is to be sent for this topic, even if no local subscriptions exist, to directly connected queue managers (parameter identifier: MQIA_PROXY_SUB).

The value can be:

MQTA_PROXY_SUB_FORCE
A proxy subscription is sent to connected queue managers even if no local subscriptions exist.

MQTA_PROXY_SUB_FIRSTUSE
A proxy subscription is sent for this topic only when a local subscription exists.

PublicationScope **(MQCFIN)**
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE).

The value can be:

MQSCOPE_ALL
Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

MQSCOPE_AS_PARENT
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQSCOPE_AS_PARENT is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Publications for this topic are not propagated to other queue managers.

Note: You can override this behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message Options.

QMgrName **(MQCFST)**
Name of local queue manager (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH

SubscriptionScope **(MQCFIN)**
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE).

The value can be:

MQSCOPE_ALL
Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

MQSCOPE_AS_PARENT
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQSCOPE_AS_PARENT is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Subscriptions for this topic are not propagated to other queue managers.

Note: You can override this behavior on a subscription-by-subscription basis, using MQSO_SCOPE_QMGR on the Subscription Descriptor or SUBSCOPE(QMGR) on DEFINE SUB.

TopicDesc **(MQCFST)**
Topic description (parameter identifier: MQCA_TOPIC_DESC).

The maximum length is MQ_TOPIC_DESC_LENGTH.

TopicName **(MQCFST)**
Topic object name (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH

TopicString **(MQCFST)**
The topic string (parameter identifier: MQCA_TOPIC_STRING).

The '/' character within this string has special meaning. It delimits the elements in the topic tree. A topic string can start with the '/' character but is not required to. A string starting with the '/' character is not the same as the string which starts without the '/' character. A topic string cannot end with the "/" character.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

TopicType **(MQCFIN)**
Whether this object is a local or cluster topic (parameter identifier: MQIA_TOPIC_TYPE).

The value can be:

MQTOPT_LOCAL
This object is a local topic.

MQTOPT_CLUSTER
This object is a cluster topic.

UseDLQ **(MQCFIN)**
Whether the dead-letter queue (or undelivered message queue) should be used when publication messages cannote be delivered to their correct subscriber queue (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value might be:

MQUSEDLQ_NO
Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message and the application's MQPUT to a topic will fail in accordance with the settings of NPMSGDLV and PMSGDLV.

MQUSEDLQ_YES
If the queue manager DEADQ attribute provides the name of a dead-letter queue then it will be used, otherwise the behvaiour will be as for MQUSEDLQ_NO.

MQUSEDLQ_AS_PARENT
Whether to use the dead-letter queue is based on the setting of the closest administrative topic object in the topic tree.

WildcardOperation **(MQCFIN)**
Behavior of subscriptions including wildcards made to this topic (parameter identifier: MQIA_WILDCARD_OPERATION).

The value can be:

MQTA_PASSTHRU
Subscriptions made using wildcard topic names that are less specific than the topic string at this topic object receive publications made to this topic and to topic strings more specific than this topic. MQTA_PASSTHRU is the default supplied with WebSphere MQ.

MQTA_BLOCK
Subscriptions made using wildcard topic names that are less specific than the topic string at this topic object do not receive publications made to this topic or to topic strings more specific than this topic.

Inquire Topic Names:

The Inquire Topic Names (MQCMD_INQUIRE_TOPIC_NAMES) command inquires a list of administrative topic names that match the generic topic name specified.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

TopicName **(MQCFST)**
Administrative topic object name (parameter identifier: MQCA_TOPIC_NAME).

Specifies the name of the administrative topic object that information is to be returned for.

Generic topic object names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
Disposition of the object within the group (parameter identifier: MQIA_QSG_DISP). This parameter applies to z/OS only.

Specifies the disposition of the object for which information is to be returned (that is, where it is defined and how it behaves). The value can be:

MQQSGD_LIVE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_LIVE is the default value if the parameter is not specified.

MQQSGD_ALL
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY.

If there is a shared queue manager environment, and the command is being executed on the queue manager where it was issued, this option also displays information for objects defined with MQQSGD_GROUP.

If MQQSGD_LIVE is specified or defaulted, or if MQQSGD_ALL is specified in a shared queue manager environment, the command might give duplicated names (with different dispositions).

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP. MQQSGD_GROUP is permitted only in a shared queue environment.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

MQQSGD_PRIVATE
The object is defined as MQQSGD_Q_MGR or MQQSGD_COPY. MQQSGD_PRIVATE returns the same information as MQQSGD_LIVE.

Inquire Topic Names (Response):

The response to the Inquire Topic Names (MQCMD_INQUIRE_TOPIC_NAMES) command consists of the response header followed by a parameter structure giving zero or more names that match the specified administrative topic name.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Additionally, on z/OS only, the *QSGDispositions* parameter structure (with the same number of entries as the *TopicNames* structure) is returned. Each entry in this structure indicates the disposition of the object with the corresponding entry in the *TopicNames* structure.

Always returned:
TopicNames, *QSGDispositions*

Returned if requested:
None

Response data

TopicNames **(MQCFSL)**
List of topic object names (parameter identifier: MQCACF_TOPIC_NAMES).

QSGDispositions **(MQCFIL)**
List of QSG dispositions (parameter identifier: MQIACF_QSG_DISPS). This parameter is valid on z/OS only. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_GROUP
The object is defined as MQQSGD_GROUP.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

Inquire Topic Status:

The Inquire Topic Status (MQCMD_INQUIRE_TOPIC_STATUS) command inquires the status of a particular topic, or of a topic and its child topics. The Inquire Topic Status command has a required parameter. The Inquire Topic Status command has optional parameters.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

TopicString **(MQCFST)**
The topic string (parameter identifier: MQCA_TOPIC_STRING).

The name of the topic string to display. WebSphere MQ uses the topic wildcard characters ('#' and '+') and does not treat a trailing asterisk as a wildcard. For more information about using wildcard characters, refer to the related topic.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

Optional parameters

StatusType **(MQCFIN)**
The type of status to return (parameter identifier: MQIACF_TOPIC_STATUS_TYPE).

The value can be:

MQIACF_TOPIC_STATUS

MQIACF_TOPIC_SUB

MQIACF_TOPIC_PUB

This command ignores any attribute selectors specified in the *TopicStatusAttrs* list that are not valid for the selected *StatusType* and the command raises no error.

The default value if this parameter is not specified is **MQIACF_TOPIC_STATUS**.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- Blank (or omit the parameter altogether). The command runs on the queue manager on which you enter it.
- A queue manager name. The command runs on the queue manager that you specify, if it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which you entered the command, you must be using a queue-sharing group environment, and the command server must be enabled.

- An asterisk (*). The command runs on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

You cannot use CommandScope as a filter parameter.

IntegerFilterCommand **(MQCFIF)**
Integer filter command descriptor that you use to restrict the output from the command. The parameter identifier must be an integer type and must be one of the values allowed for *MQIACF_TOPIC_SUB_STATUS*, *MQIACF_TOPIC_PUB_STATUS* or *MQIACF_TOPIC_STATUS*, except *MQIACF_ALL*.

If you specify an integer filter, you cannot also specify a string filter with the *StringFilterCommand* parameter.

StringFilterCommand **(MQCFSF)**
String filter command descriptor. The parameter identifier must be any string type parameter allowed for *MQIACF_TOPIC_SUB_STATUS*, *MQIACF_TOPIC_PUB_STATUS* or *MQIACF_TOPIC_STATUS*, except *MQIACF_ALL*, or the identifier *MQCA_TOPIC_STRING_FILTER* to filter on the topic string.

Use the parameter identifier to restrict the output from the command by specifying a filter condition. Ensure that the parameter is valid for the type selected in StatusType. If you specify a string filter, you cannot also specify an integer filter using the *IntegerFilterCommand* parameter.

TopicStatusAttrs **(MQCFIL)**
Topic status attributes (parameter identifier: MQIACF_TOPIC_STATUS_ATTRS)

The default value used if the parameter is not specified is:

MQIACF_ALL

You can specify any of the parameter values listed in the related reference about Response Data. It is not an error to request status information that is not relevant for a particular status type, but the response contains no information for the value concerned.

Inquire Topic Status (Response):

The response of the Inquire topic (MQCMD_INQUIRE_TOPIC_STATUS) command consists of the response header, followed by the *TopicString* structure, and the requested combination of attribute parameter structures (where applicable). The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS. The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS_SUB. The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS_PUB.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Always returned:
TopicString

Returned if requested and StatusType is MQIACF_TOPIC_STATUS:
Cluster, *DefPriority*, *DefaultPutResponse*, *DefPersistence*, *DurableSubscriptions*, *InhibitPublications*, *InhibitSubscriptions*, *AdminTopicName*, *DurableModelQName*, *NonDurableModelQName*, *PersistentMessageDelivery*, *NonPersistentMessageDelivery*, *RetainedPublication*, *PublishCount*, *SubscriptionScope*, *SubscriptionCount*, *PublicationScope*, *UseDLQ*

Note: The Inquire Topic Status command returns only resolved values for the topic, and no AS_PARENT values.

Returned if requested and StatusType is MQIACF_TOPIC_SUB:
SubscriptionId, *SubscriptionUserId*, *Durable*, *SubscriptionType*, *ResumeDate*, *ResumeTime*, *LastMessageDate*, *LastMessageTime*, *NumberOfMessages*, *ActiveConnection*

Returned if requested and StatusType is MQIACF_TOPIC_PUB:
LastPublishDate, *LastPublishTime*, *NumberOfPublishes*, *ActiveConnection*

Response data (TOPIC_STATUS)

ClusterName **(MQCFST)**
The name of the cluster to which this topic belongs (parameter identifier: MQCA_CLUSTER_NAME).

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

The value can be as follows:

Blank This topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers.

Blank is the default value for this parameter if no value is specified.

String This topic belongs to the indicated cluster.

Additionally, if PublicationScope or SubscriptionScope is set to MQSCOPE_ALL, this cluster is to be used for the propagation of publications and subscriptions, for this topic, to publish/subcribe cluster-connected queue managers.

DefPersistence **(MQCFIN)**
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE).

Returned value:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefaultPutResponse **(MQCFIN)**
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

Returned value:

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

DefPriority **(MQCFIN)**
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Shows the resolved default priority of messages published to the topic.

DurableSubscriptions **(MQCFIN)**
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB).

Returned value:

MQSUB_DURABLE_ALLOWED
Durable subscriptions are permitted.

MQSUB_DURABLE_INHIBITED
Durable subscriptions are not permitted.

InhibitPublications **(MQCFIN)**
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB).

Returned value:

MQTA_PUB_INHIBITED
Publications are inhibited for this topic.

MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions **(MQCFIN)**
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB).

Returned value:

MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.

MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

AdminTopicName **(MQCFST)**
Topic object name (parameter identifier: MQCA_ADMIN_TOPIC_NAME).

If the topic is an admin-node, the command displays the associated topic object name containing the node configuration. If the field is not an admin-node the command displays a blank.

The maximum length of the string is MQ_TOPIC_NAME_LENGTH.

DurableModelQName **(MQCFST)**
The name of the model queue used for managed durable subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

Shows the resolved value of the name of the model queue to be used for durable subscriptions that request the queue manager to manage the destination of publications.

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonDurableModelQName **(MQCFST)**
The name of the model queue for managed non-durable subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

PersistentMessageDelivery **(MQCFIN)**
Delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY).

Returned value:

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability, for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

NonPersistentMessageDelivery **(MQCFIN)**
Delivery mechanism for non-persistent messages published to this topic (parameter identifier: MQIA_NPM_DELIVERY).

Returned value:

MQDLV_ALL
Non-persistent messages must be delivered to all subscribers, irrespective of durability, for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_DUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no subscribers receive the message and the MQPUT call fails.

MQDLV_ALL_AVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

RetainedPublication **(MQCFIN)**
Whether there is a retained publication for this topic (parameter identifier: MQIACF_RETAINED_PUBLICATION).

Returned value:

MQQSO_YES
There is a retained publication for this topic.

MQQSO_NO
There is no retained publication for this topic.

PublishCount **(MQCFIN)**
Publish count (parameter identifier: MQIA_PUB_COUNT).

The number of applications currently publishing to the topic.

SubscriptionCount **(MQCFIN)**
Subscription count (parameter identifier: MQIA_SUB_COUNT).

The number of subscribers for this topic string, including durable subscribers who are not currently connected.

SubscriptionScope **(MQCFIN)**
Determines whether this queue manager propagates subscriptions for this topic to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE).

Returned value:

MQSCOPE_QMGR
The queue manager does not propagate subscriptions for this topic to other queue managers.

MQSCOPE_ALL
The queue manager propagates subscriptions for this topic to hierarchically connected queue managers and to publish/subscribe cluster connected queues.

PublicationScope **(MQCFIN)**
Determines whether this queue manager propagates publications for this topic to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE).

Returned value:

MQSCOPE_QMGR
The queue manager does not propagate publications for this topic to other queue managers.

MQSCOPE_ALL
The queue manager propagates publications for this topic to hierarchically connected queue managers and to publish/subscribe cluster connected queues.

UseDLQ **(MQCFIN)**
Determines whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue (parameter identifier: MQIA_USE_DEAD_LETTER_Q).

The value can be:

MQUSEDLQ_NO
Publication messages that cannot be delivered to their correct subscriber queue are treated as a failure to put the message. The MQPUT of an application to a topic fails in accordance with the settings of MQIA_NPM_DELIVERY and MQIA_PM_DELIVERY.

MQUSEDLQ_YES
If the DEADQ queue manager attribute provides the name of a dead-letter queue then it is used, otherwise the behavior is as for MQUSEDLQ_NO.

Response data (TOPIC_STATUS_SUB)

SubscriptionId **(MQCFBS)**
Subscription identifier (parameter identifier: MQBACF_SUB_ID).

The queue manager assigns *SubscriptionId* as an all time unique identifier for this subscription.

The maximum length of the string is MQ_CORREL_ID_LENGTH.

SubscriptionUserId **(MQCFST)**
The user ID that owns this subscription (parameter identifier: MQCACF_SUB_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

Durable **(MQCFIN)**
Whether this subscription is a durable subscription (parameter identifier: MQIACF_DURABLE_SUBSCRIPTION).

MQSUB_DURABLE_YES
The subscription persists, even if the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. The queue manager reinstates the subscription during restart.

MQSUB_DURABLE_NO
The subscription is non-durable. The queue manager removes the subscription when the creating application disconnects from the queue manager or issues an MQCLOSE call for the subscription. If the subscription has a destination class (DESTCLAS) of MANAGED, the queue manager removes any messages not yet consumed when it closes the subscription.

SubscriptionType **(MQCFIN)**
The type of subscription (parameter identifier: MQIACF_SUB_TYPE).

The value can be:

MQSUBTYPE_ADMIN

MQSUBTYPE_API

MQSUBTYPE_PROXY

ResumeDate **(MQCFST)**
Date of the most recent MQSUB call that connected to this subscription (parameter identifier: MQCA_RESUME_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

ResumeTime **(MQCFST)**
Time of the most recent MQSUB call that connected to this subscription (parameter identifier: MQCA_RESUME_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

LastMessageDate **(MQCFST)**
Date on which an MQPUT call last sent a message to this subscription. The queue manager updates the date field after the MQPUT call successfully puts a message to the destination specified by this subscription (parameter identifier: MQCACF_LAST_MSG_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

Note: An **MQSUBRQ** call updates this value.

LastMessageTime **(MQCFST)**
Time at which an MQPUT call last sent a message to this subscription. The queue manager updates the time field after the MQPUT call successfully puts a message to the destination specified by this subscription (parameter identifier: MQCACF_LAST_MSG_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

Note: An **MQSUBRQ** call updates this value.

NumberOfMessages **(MQCFIN)**
Number of messages put to the destination specified by this subscription (parameter identifier: MQIACF_MESSAGE_COUNT).

Note: An **MQSUBRQ** call updates this value.

ActiveConnection **(MQCFBS)**
The currently active *ConnectionId* (CONNID) that opened this subscription (parameter identifier: MQBACF_CONNECTION_ID).

The maximum length of the string is MQ_CONNECTION_ID_LENGTH.

Response data (TOPIC_STATUS_PUB)

LastPublicationDate **(MQCFST)**
Date on which this publisher last sent a message (parameter identifier: MQCACF_LAST_PUB_DATE).

The maximum length of the string is MQ_DATE_LENGTH.

LastPublicationTime **(MQCFST)**
Time at which this publisher last sent a message (parameter identifier: MQCACF_LAST_PUB_TIME).

The maximum length of the string is MQ_TIME_LENGTH.

NumberOfPublishes **(MQCFIN)**
Number of publishes made by this publisher (parameter identifier: MQIACF_PUBLISH_COUNT).

ActiveConnection **(MQCFBS)**
The currently active *ConnectionId* (CONNID) associated with the handle that has this topic open for publish (parameter identifier: MQBACF_CONNECTION_ID).

The maximum length of the string is MQ_CONNECTION_ID_LENGTH.

Ping Channel:

The Ping Channel (MQCMD_PING_CHANNEL) command tests a channel by sending data as a special message to the remote message queue manager and checking that the data is returned. The data is generated by the local queue manager.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

This command can only be used for channels with a *ChannelType* value of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_CLUSSDR.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

The command is not valid if the channel is running; however it is valid if the channel is stopped or in retry mode.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be tested. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

DataCount **(MQCFIN)**
Data count (parameter identifier: MQIACH_DATA_COUNT).

Specifies the length of the data.

Specify a value in the range 16 through 32 768. The default value is 64 bytes.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ChannelDisposition **(MQCFIN)**
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be tested.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

MQCHLD_FIXSHARED
Tests shared channels, tied to a specific queue manager.

The combination of the *ChannelDisposition* and *CommandScope* parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 101

Table 101. ChannelDisposition and CommandScope for PING CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_PRIVATE	Ping private channel on the local queue manager	Ping private channel on the named queue manager	Ping private channel on all active queue managers

Table 101. ChannelDisposition and CommandScope for PING CHANNEL (continued)

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_SHARED	Ping a shared channel on the most suitable queue manager in the group MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted
MQCHLD_FIXSHARED	Ping a shared channel on the local queue manager	Ping a shared channel on the named queue manager	Not permitted

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_ALLOCATE_FAILED
Allocation failed.

MQRCCF_BIND_FAILED
Bind failed.

MQRCCF_CCSID_ERROR
Coded character-set identifier error.

MQRCCF_CHANNEL_CLOSED
Channel closed.

MQRCCF_CHANNEL_IN_USE
Channel in use.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CONFIGURATION_ERROR
Configuration error.

MQRCCF_CONNECTION_CLOSED
Connection closed.

MQRCCF_CONNECTION_REFUSED
Connection refused.

MQRCCF_DATA_TOO_LARGE
Data too large.

MQRCCF_ENTRY_ERROR
Connection name not valid.

MQRCCF_HOST_NOT_AVAILABLE
Remote system not available.

MQRCCF_NO_COMMS_MANAGER
Communications manager not available.

MQRCCF_PING_DATA_COMPARE_ERROR
Ping Channel command failed.

MQRCCF_PING_DATA_COUNT_ERROR
Data count not valid.

MQRCCF_PING_ERROR
Ping error.

MQRCCF_RECEIVE_FAILED
Receive failed.

MQRCCF_RECEIVED_DATA_ERROR
Received data error.

MQRCCF_REMOTE_QM_TERMINATING
Remote queue manager terminating.

MQRCCF_REMOTE_QM_UNAVAILABLE
Remote queue manager not available.

MQRCCF_SEND_FAILED
Send failed.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_TERMINATED_BY_SEC_EXIT
Channel terminated by security exit.

MQRCCF_UNKNOWN_REMOTE_CHANNEL
Remote channel not known.

MQRCCF_USER_EXIT_NOT_AVAILABLE
User exit not available.

Ping Queue Manager:

The Ping Queue Manager (MQCMD_PING_Q_MGR) command tests whether the queue manager and its command server is responsive to commands. If the queue manager is responding a positive reply is returned.

HP Integrity NonStop Server	UNIX and Linux systems	Windows
X	X	X

Required parameters:
None

Optional parameters:
None

Purge Channel:

The Purge Channel (MQCMD_PURGE_CHANNEL) command stops and purges a IBM WebSphere MQ telemetry channel.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

This command can only be issued to an MQTT channel type.

Purging a telemetry channel disconnects all the MQTT clients connect to it, cleans up the state of the MQTT clients, and stops the telemetry channel. Cleaning the state of a client deletes all the pending publications and removes all the subscriptions from the client.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be stopped and purged. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType **(MQCFIN)**
Channel type. This parameter must follow immediately after the **ChannelName** parameter on all platforms except z/OS, and the value must be MQTT.

Optional parameters

ClientIdentifier **(MQCFST)**
Client identifier. The client identifier is a 23-byte string that identifies a IBM WebSphere MQ Telemetry Transport client. When the Purge Channel command specifies a *ClientIdentifier*, only the connection for the specified client identifier is purged. If the *ClientIdentifier* is not specified, all the connections on the channel are purged.

The maximum length of the string is MQ_CLIENT_ID_LENGTH.

Refresh Cluster:

The Refresh Cluster (MQCMD_REFRESH_CLUSTER) command discards all locally held cluster information, including any auto-defined channels that are not in doubt, and forces the repository to be rebuilt.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

Required parameters

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to be refreshed.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

This parameter is the name of the cluster to be refreshed. If an asterisk (*) is specified for the name, the queue manager is refreshed in all the clusters to which it belongs.

If an asterisk (*) is specified with *RefreshRepository* set to MQCFO_REFRESH_REPOSITORY_YES, the queue manager restarts its search for repository queue managers, using information in the local cluster-sender channel definitions.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

RefreshRepository **(MQCFIN)**
Whether repository information is refreshed (parameter identifier: MQIACF_REFRESH_REPOSITORY).

This parameter indicates whether the information about repository queue managers is refreshed.

The value can be:

MQCFO_REFRESH_REPOSITORY_YES
Refresh repository information.

This value cannot be specified if the queue manager is itself a repository queue manager.

MQCFO_REFRESH_REPOSITORY_YES specifies that in addition to MQCFO_REFRESH_REPOSITORY_NO behavior, objects representing full repository cluster queue managers are also refreshed. Do not use this option if the queue manager is itself a full repository.

If it is a full repository, you must first alter it so that it is not a full repository for the cluster in question.

The full repository location is recovered from the manually defined cluster-sender channel definitions. After the refresh with MQCFO_REFRESH_REPOSITORY_YES has been issued the queue manager can be altered so that it is once again a full repository.

MQCFO_REFRESH_REPOSITORY
Do not refresh repository information. MQCFO_REFRESH_REPOSITORY is the default.

If you select MQCFO_REFRESH_REPOSITORY_YES, check that all cluster-sender channels in the relevant cluster are inactive or stopped before you issue the Refresh Cluster command. If there are cluster-sender channels running at the time when the Refresh is processed, and they are used exclusively by the cluster or clusters being refreshed and MQCFO_REFRESH_REPOSITORY_YES is used, the channels are stopped, by using the Stop Channel command with a value of MQMODE_FORCE in the *Mode* parameter if necessary.

This scenario ensures that the Refresh can remove the channel state and that the channel will run with the refreshed version after the Refresh has completed. If the state of a channel cannot be deleted, for example because it is in doubt, or because it is also running as part of another cluster, it is state is not new after the refresh and it does not automatically restart if it was stopped.

Related information:

Clustering: Using REFRESH CLUSTER best practices

Refresh Queue Manager:

Use the Refresh Queue Manager (MQCMD_REFRESH_Q_MGR) command to perform special operations on queue managers.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

RefreshType **(MQCFIN)**
Type of information to be refreshed (parameter identifier: MQIACF_REFRESH_TYPE).

Use this parameter to specify the type of information to be refreshed. The value can be:

MQRT_CONFIGURATION
MQRT_CONFIGURATION causes the queue manager to generate configuration event messages for every object definition that matches the selection criteria specified by the *ObjectType*, *ObjectName*, and *RefreshInterval* parameters.

A Refresh Queue Manager command with a *RefreshType* value of MQRT_CONFIGURATION is generated automatically when the value of the queue manager's *ConfigurationEvent* parameter changes from MQEVR_DISABLED to MQEVR_ENABLED.

Use this command with a *RefreshType* of MQRT_CONFIGURATION to recover from problems such as errors on the event queue. In such cases, use appropriate selection criteria, to avoid excessive processing time and event message generation.

MQRT_EXPIRY
This requests that the queue manager performs a scan to discard expired messages for every queue that matches the selection criteria specified by the *ObjectName* parameter.

Note: Valid only on z/OS.

MQRT_PROXYSUB
Requests that the queue manager resynchronizes the proxy subscriptions that are held with and on behalf of queue managers that are connected in a hierarchy or a publish/subscribe cluster.

You must resynchronize the proxy subscriptions only in exceptional circumstances, for example, when the queue manager is receiving subscriptions that it must not be sent, or not receiving subscriptions that it must receive. The following list describes some of the exceptional reasons for resynchronizing proxy subscriptions:

- Disaster recovery.
- Problems that are identified in a queue manager error log where messages inform of the issuing of the REFRESH QMGR TYPE(REPOS) command.
- Operator errors, for example, issuing a DELETE SUB command on a proxy subscription.

Missing proxy subscriptions can be caused if the closest matching topic definition is specified with **Subscription scope** set to `Queue Manager` or it has an empty or incorrect cluster name. Note that **Publication scope** does not prevent the sending of proxy subscriptions, but does prevent publications from being delivered to them.

Extraneous proxy subscriptions can be caused if the closest matching topic definition is specified with **Proxy subscription behavior** set to `Force`.

Missing or extraneous proxy subscriptions that are due to configuration errors are not changed by issuing a resynchronization. A resynchronization does resolve missing or extraneous publications as a result of the exceptional reasons listed.

Optional parameters (Refresh Queue Manager)

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ObjectName **(MQCFST)**
Name of object to be included in the processing of this command (parameter identifier: MQCACF_OBJECT_NAME).

Use this parameter to specify the name of the object to be included in the processing of this command.

Generic names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length is MQ_OBJECT_NAME_LENGTH.

ObjectType **(MQCFIN)**
Object type for which configuration data is to be refreshed (parameter identifier: MQIACF_OBJECT_TYPE).

Use this parameter to specify the object type for which configuration data is to be refreshed. This parameter is valid only if the value of *RefreshType* is MQRT_CONFIGURATION. The default value, in that case, is MQOT_ALL. The value can be one of:

MQOT_AUTH_INFO
Authentication information object.

MQOT_CF_STRUC
CF structure.

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_LOCAL_Q
Local queue.

MQOT_MODEL_Q
Model queue.

MQOT_ALIAS_Q
Alias queue.

MQOT_REMOTE_Q
Remote queue.

MQOT_Q_MGR
Queue manager.

MQOT_CFSTRUC
CF structure.

MQOT_SERVICE
Service.

Note: Not valid on z/OS.

MQOT_STORAGE_CLASS
Storage class.

MQOT_TOPIC
Topic name.

RefreshInterval **(MQCFIN)**
Refresh interval (parameter identifier: MQIACF_REFRESH_INTERVAL).

Use this parameter to specify a value, in minutes, defining a period immediately before the current time. This requests that only objects that have been created or altered within that period (as defined by their *AlterationDate* and *AlterationTime* attributes) are included.

Specify a value in the range zero through 999 999. A value of zero means there is no time limit (0 is the default).

This parameter is valid only if the value of *RefreshType* is MQRT_CONFIGURATION.

Refresh Security:

The Refresh Security (MQCMD_REFRESH_SECURITY) command refreshes the list of authorizations held internally by the authorization service component.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

SecurityItem **(MQCFIN)**
Resource class for which the security refresh is to be performed (parameter identifier: MQIACF_SECURITY_ITEM). This parameter applies to z/OS only.

Use this parameter to specify the resource class for which the security refresh is to be performed. The value can be:

MQSECITEM_ALL
A full refresh of the type specified is performed. MQSECITEM_ALL is the default value.

MQSECITEM_MQADMIN
Specifies that administration type resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MQNLIST
Specifies that namelist resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MQPROC
Specifies that process resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MQQUEUE
Specifies that queue resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXADMIN
Specifies that administration type resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXNLIST
Specifies that namelist resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXPROC
Specifies that process resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXQUEUE
Specifies that queue resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

MQSECITEM_MXTOPIC
Specifies that topic resources are to be refreshed. Valid only if the value of *SecurityType* is MQSECTYPE_CLASSES.

SecurityType **(MQCFIN)**
Security type (parameter identifier: MQIACF_SECURITY_TYPE).

Use this parameter to specify the type of security refresh to be performed. The value can be:

MQSECTYPE_AUTHSERV
The list of authorizations held internally by the authorization services component is refreshed. MQSECTYPE_AUTHSERV is not valid on z/OS.

MQSECTYPE_AUTHSERV is the default on platforms other than z/OS.

MQSECTYPE_CLASSES
Permits you to select specific resource classes for which to perform the security refresh.

MQSECTYPE_CLASSES is valid only on z/OS where it is the default.

MQSECTYPE_SSL
MQSECTYPE_SSL refreshes the locations of the LDAP servers to be used for Certified Revocation Lists and the key repository. It also refreshes any cryptographic hardware parameters specified through WebSphere MQ and the cached view of the Secure Sockets Layer key repository. It also allows updates to become effective on successful completion of the command.

MQSECTYPE_SSL updates all SSL channels currently running, as follows:

- Sender, server, and cluster-sender channels using SSL are allowed to complete the current batch. In general, they then run the SSL handshake again with the refreshed view of the SSL key repository. However, you must manually restart a requester-server channel on which the server definition has no CONNAME parameter.
- All other channel types using SSL are stopped with a STOP CHANNEL MODE(FORCE) STATUS(INACTIVE) command. If the partner end of the stopped message channel has retry values defined, the channel tries again and the new SSL handshake uses the refreshed view of the contents of the SSL key repository, the location of the LDAP server to be used for Certification Revocation Lists, and the location of the key repository. If there is a server-connection channel, the client application loses its connection to the queue manager and must reconnect in order to continue.

Reset Channel:

The Reset Channel (MQCMD_RESET_CHANNEL) command resets the message sequence number for a WebSphere MQ channel with, optionally, a specified sequence number to be used the next time that the channel is started.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

This command can be issued to a channel of any type (except MQCHT_SVRCONN and MQCHT_CLNTCONN). However, if it is issued to a sender (MQCHT_SENDER), server (MQCHT_SERVER), or cluster-sender (MQCHT_CLUSSDR) channel, the value at both ends (issuing end and receiver or requester end), is reset when the channel is next initiated or resynchronized. The value at both ends is reset to be equal.

If the command is issued to a receiver (MQCHT_RECEIVER), requester (MQCHT_REQUESTER), or cluster-receiver (MQCHT_CLUSRCVR) channel, the value at the other end is *not* reset as well; this step must be done separately if necessary.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be reset. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

ChannelDisposition **(MQCFIN)**
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be reset.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

The combination of the *ChannelDisposition* and *CommandScope* parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 102

Table 102. ChannelDisposition and CommandScope for RESET CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**
MQCHLD_PRIVATE	Reset private channel on the local queue manager	Reset private channel on the named queue manager
MQCHLD_SHARED	Reset a shared channel on all active queue managers. MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted

MsgSeqNumber **(MQCFIN)**
Message sequence number (parameter identifier: MQIACH_MSG_SEQUENCE_NUMBER).

Specifies the new message sequence number.

The value must be in the range 1 through 999 999 999. The default value is one.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

Reset Cluster:

The Reset Cluster (MQCMD_RESET_CLUSTER) command forces a queue manager to leave a cluster.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to be reset.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

QMgrIdentifier **(MQCFST)**
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

This parameter is the unique identifier of the queue manager to be forcibly removed from the cluster. Only one of `QMgrIdentifier` and `QMgrName` can be specified. Use `QMgrIdentifier` in preference to `QmgrName`, because `QmgrName` might not be unique.

QMgrName **(MQCFST)**
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

This parameter is the name of the queue manager to be forcibly removed from the cluster. Only one of `QMgrIdentifier` and `QMgrName` can be specified. Use `QMgrIdentifier` in preference to `QmgrName`, because `QmgrName` might not be unique.

Action **(MQCFIN)**
Action (parameter identifier: MQIACF_ACTION).

Specifies the action to take place. This parameter can be requested only by a repository queue manager.

The value can be:

MQACT_FORCE_REMOVE
Requests that a queue manager is forcibly removed from a cluster.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

RemoveQueues **(MQCFIN)**
Whether cluster queues are removed from the cluster (parameter identifier: MQIACF_REMOVE_QUEUES).

This parameter indicates whether the cluster queues that belong to the queue manager being removed from the cluster are to be removed from the cluster. This parameter can be specified even if the queue manager identified by the *QMgrName* parameter is not currently in the cluster.

The value can be:

MQCFO_REMOVE_QUEUES_YES
Remove queues belonging to the queue manager being removed from the cluster.

MQCFO_REMOVE_QUEUES_NO
Do not remove queues belonging to the queue manager being removed. MQCFO_REMOVE_QUEUES_NO is the default.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_ACTION_VALUE_ERROR
Value not valid.

Reset Queue Manager:

Use the Reset Queue Manager (MQCMD_RESET_Q_MGR) command as part of your backup and recovery procedures on AIX, HP-UX, Linux, Solaris, IBM i, and Windows.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

You can use this command to request that the queue manager starts writing to a new log extent, making the previous log extent available for archiving.

Use the Reset Queue Manager (MQCMD_RESET_Q_MGR) command to forcibly remove a publish/subscribe hierarchical connection for which this queue manager is nominated as either the parent or the child in a hierarchical connection. Valid on all supported platforms.

Required parameters

Action **(MQCFIN)**
Action (parameter identifier: MQIACF_ACTION).

Specifies the action to take place.

The value can be:

MQACT_ADVANCE_LOG
Requests that the queue manager starts writing to a new log extent, making the previous log extent available for archiving. This command is accepted only if the queue manager is configured to use linear logging.

Note: Not valid on Compaq NSK or z/OS.

MQACT_COLLECT_STATISTICS
Requests that the queue manager ends the current statistics collection period, and writes the statistics collected.

Note: Not valid on Compaq NSK, or z/OS.

MQACT_PUBSUB
Requests a publish/subscribe reset. This value requires that one of the optional parameters, ChildName or ParentName, is specified.

Optional parameters

ChildName **(MQCFST)**
The name of the child queue manager for which the hierarchical connection is to be forcibly canceled (parameter identifier: MQCA_CHILD).

This attribute is valid only when the Action parameter has the value MQACT_PUBSUB.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ParentName **(MQCFST)**
The name of the parent queue manager for which the hierarchical connection is to be forcibly canceled (parameter identifier: MQCA_PARENT).

This attribute is valid only when the Action parameter has the value MQACT_PUBSUB.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRC_RESOURCE_PROBLEM
Insufficient system resources available.

Reset Queue Statistics:

The Reset Queue Statistics (MQCMD_RESET_Q_STATS) command reports the performance data for a queue and then resets the performance data. Performance data is maintained for each local queue (including transmission queues).

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Performance data is reset at the following times:

- When a Reset Queue Statistics command is issued
- When the queue manager is restarted
- When a performance event is generated for a queue

Required parameters

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the local queue to be tested and reset.

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_Q_WRONG_TYPE
Action not valid for the queue of specified type.

MQRCCF_EVENTS_DISABLED
The queue manager performance events are disabled (PERFMEV). On z/OS, it is necessary to enable queue manager performance events to use this command. For more details, see the PerformanceEvent property in the "Change Queue Manager" on page 882 command.

Reset Queue Statistics (Response):

The response to the Reset Queue Statistics (MQCMD_RESET_Q_STATS) command consists of the response header followed by the *QName* structure and the attribute parameter structures shown in the following sections.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

If a generic queue name was specified, one such message is generated for each queue found.

Always returned:
HighQDepth, *MsgDeqCount*, *MsgEnqCount*, *QName*, *QSGDisposition*, *TimeSinceReset*

Response data

HighQDepth **(MQCFIN)**
Maximum number of messages on a queue (parameter identifier: MQIA_HIGH_Q_DEPTH).

This count is the peak value of the *CurrentQDepth* local queue attribute since the last reset. The *CurrentQDepth* is incremented during an MQPUT call, and during backout of an MQGET call, and is decremented during a (nonbrowse) MQGET call, and during backout of an MQPUT call.

MsgDeqCount **(MQCFIN)**
Number of messages dequeued (parameter identifier: MQIA_MSG_DEQ_COUNT).

This count includes messages that have been successfully retrieved (with a nonbrowse MQGET) from the queue, even though the MQGET has not yet been committed. The count is not decremented if the MQGET is later backed out.

On z/OS, if the value exceeds 999 999 999, it is returned as 999 999 999

MsgEnqCount **(MQCFIN)**
Number of messages enqueued (parameter identifier: MQIA_MSG_ENQ_COUNT).

This count includes messages that have been put to the queue, but have not yet been committed. The count is not decremented if the put is later backed out.

On z/OS, if the value exceeds 999 999 999, it is returned as 999 999 999

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QSGDisposition **(MQCFIN)**
QSG disposition (parameter identifier: MQIA_QSG_DISP).

Specifies the disposition of the object (that is, where it is defined and how it behaves). This parameter is valid on z/OS only. The value can be:

MQQSGD_COPY
The object is defined as MQQSGD_COPY.

MQQSGD_SHARED
The object is defined as MQQSGD_SHARED.

MQQSGD_Q_MGR
The object is defined as MQQSGD_Q_MGR.

TimeSinceReset **(MQCFIN)**
Time since statistics reset in seconds (parameter identifier: MQIA_TIME_SINCE_RESET).

Resolve Channel:

The Resolve Channel (MQCMD_RESOLVE_CHANNEL) command requests a channel to commit or back out in-doubt messages. This command is used when the other end of a link fails during the confirmation stage, and for some reason it is not possible to reestablish the connection. In this situation the sending end remains in an in-doubt state, whether the messages were received. Any outstanding units of work must be resolved using Resolve Channel with either backout or commit.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Care must be exercised in the use of this command. If the resolution specified is not the same as the resolution at the receiving end, messages can be lost or duplicated.

This command can only be used for channels with a *ChannelType* value of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_CLUSSDR.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be resolved. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

InDoubt **(MQCFIN)**
Indoubt resolution (parameter identifier: MQIACH_IN_DOUBT).

Specifies whether to commit or back out the in-doubt messages.

The value can be:

MQIDO_COMMIT
Commit.

MQIDO_BACKOUT
Backout.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

ChannelDisposition **(MQCFIN)**
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be resolved.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

The combination of the *ChannelDisposition* and *CommandScope* parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 103

Table 103. ChannelDisposition and CommandScope for RESOLVE CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**
MQCHLD_PRIVATE	Resolve private channel on the local queue manager	Resolve private channel on the named queue manager
MQCHLD_SHARED	Resolve a shared channel on all active queue managers. MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_INDOUBT_VALUE_ERROR
In-doubt value not valid.

Resume Queue Manager Cluster:

The Resume Queue Manager Cluster (MQCMD_RESUME_Q_MGR_CLUSTER) command informs other queue managers in a cluster that the local queue manager is again available for processing, and can be sent messages. It reverses the action of the Suspend Queue Manager Cluster (MQCMD_SUSPEND_Q_MGR_CLUSTER) command.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster for which availability is to be resumed.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist **(MQCFST)**
Cluster Namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist specifying a list of clusters for which availability is to be resumed.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

Set Authority Record:

The Set Authority Record (MQCMD_SET_AUTH_REC) command sets the authorizations of a profile, object, or class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ProfileName **(MQCFST)**
Profile name (parameter identifier: MQCACF_AUTH_PROFILE_NAME).

The authorizations apply to all WebSphere MQ objects with names that match the profile name specified. You can define a generic profile. If you specify an explicit profile name, the object must exist.

The maximum length of the string is MQ_AUTH_PROFILE_NAME_LENGTH.

ObjectType **(MQCFIN)**
The type of object for which to set authorizations (parameter identifier: MQIACF_OBJECT_TYPE).

The value can be:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel object.

MQOT_CLNTCONN_CHANNEL
Client-connection channel object.

MQOT_COMM_INFO
Communication information object

MQOT_LISTENER
Listener object.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue, or queues, that match the object name parameter.

MQOT_Q_MGR
Queue manager.

MQOT_REMOTE_Q_MGR_NAME
Remote queue manager.

MQOT_SERVICE
Service object.

MQOT_TOPIC
Topic object.

Note: The required parameters must be in the order **ProfileName** followed by **ObjectType**.

Optional parameters

AuthorityAdd **(MQCFIL)**
Authority values to set (parameter identifier: MQIACF_AUTH_ADD_AUTHS).

This parameter is a list of authority values to set for the named profile. The values can be:

MQAUTH_NONE
The entity has authority set to 'none'.

MQAUTH_ALT_USER_AUTHORITY
Specify an alternate user ID on an MQI call.

MQAUTH_BROWSE
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

MQAUTH_CHANGE
Change the attributes of the specified object, using the appropriate command set.

MQAUTH_CLEAR
Clear a queue.

MQAUTH_CONNECT
Connect the application to the specified queue manager by issuing an MQCONN call.

MQAUTH_CREATE
Create objects of the specified type using the appropriate command set.

MQAUTH_DELETE
Delete the specified object using the appropriate command set.

MQAUTH_DISPLAY
Display the attributes of the specified object using the appropriate command set.

MQAUTH_INPUT
Retrieve a message from a queue by issuing an MQGET call.

MQAUTH_INQUIRE
Make an inquiry on a specific queue by issuing an MQINQ call.

MQAUTH_OUTPUT
Put a message on a specific queue by issuing an MQPUT call.

MQAUTH_PASS_ALL_CONTEXT
Pass all context.

MQAUTH_PASS_IDENTITY_CONTEXT
Pass the identity context.

MQAUTH_SET
Set attributes on a queue from the MQI by issuing an MQSET call.

MQAUTH_SET_ALL_CONTEXT
Set all context on a queue.

MQAUTH_SET_IDENTITY_CONTEXT
Set the identity context on a queue.

MQAUTH_CONTROL
For listeners and services, start and stop the specified channel, listener, or service.

For channels, start, stop, and ping the specified channel.

For topics, define, alter, or delete subscriptions.

MQAUTH_CONTROL_EXTENDED
Reset or resolve the specified channel.

MQAUTH_PUBLISH
Publish to the specified topic.

MQAUTH_SUBSCRIBE
Subscribe to the specified topic.

MQAUTH_RESUME
Resume a subscription to the specified topic.

MQAUTH_SYSTEM
Use queue manager for internal system operations.

MQAUTH_ALL
Use all operations applicable to the object.

MQAUTH_ALL_ADMIN
Use all administration operations applicable to the object.

MQAUTH_ALL_MQI
Use all MQI calls applicable to the object.

The contents of the *AuthorityAdd* and *AuthorityRemove* lists must be mutually exclusive. You must specify a value for either *AuthorityAdd* or *AuthorityRemove*. An error occurs if you do not specify either.

AuthorityRemove **(MQCFIL)**
Authority values to remove (parameter identifier: MQIACF_AUTH_REMOVE_AUTHS).

This parameter is a list of authority values to remove from the named profile. The values can be:

MQAUTH_NONE
The entity has authority set to 'none'.

MQAUTH_ALT_USER_AUTHORITY
Specify an alternate user ID on an MQI call.

MQAUTH_BROWSE
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

MQAUTH_CHANGE
Change the attributes of the specified object, using the appropriate command set.

MQAUTH_CLEAR
Clear a queue.

MQAUTH_CONNECT
Connect the application to the specified queue manager by issuing an MQCONN call.

MQAUTH_CREATE
Create objects of the specified type using the appropriate command set.

MQAUTH_DELETE
Delete the specified object using the appropriate command set.

MQAUTH_DISPLAY
Display the attributes of the specified object using the appropriate command set.

MQAUTH_INPUT
Retrieve a message from a queue by issuing an MQGET call.

MQAUTH_INQUIRE
Make an inquiry on a specific queue by issuing an MQINQ call.

MQAUTH_OUTPUT
Put a message on a specific queue by issuing an MQPUT call.

MQAUTH_PASS_ALL_CONTEXT
Pass all context.

MQAUTH_PASS_IDENTITY_CONTEXT
Pass the identity context.

MQAUTH_SET
Set attributes on a queue from the MQI by issuing an MQSET call.

MQAUTH_SET_ALL_CONTEXT
Set all context on a queue.

MQAUTH_SET_IDENTITY_CONTEXT
Set the identity context on a queue.

MQAUTH_CONTROL
For listeners and services, start and stop the specified channel, listener, or service.

For channels, start, stop, and ping the specified channel.

For topics, define, alter, or delete subscriptions.

MQAUTH_CONTROL_EXTENDED
Reset or resolve the specified channel.

MQAUTH_PUBLISH
Publish to the specified topic.

MQAUTH_SUBSCRIBE
Subscribe to the specified topic.

MQAUTH_RESUME
Resume a subscription to the specified topic.

MQAUTH_SYSTEM
Use queue manager for internal system operations.

MQAUTH_ALL
Use all operations applicable to the object.

MQAUTH_ALL_ADMIN
Use all administration operations applicable to the object.

MQAUTH_ALL_MQI
Use all MQI calls applicable to the object.

The contents of the *AuthorityAdd* and *AuthorityRemove* lists must be mutually exclusive. You must specify a value for either *AuthorityAdd* or *AuthorityRemove*. An error occurs if you do not specify either.

GroupNames **(MQCFSL)**
Group names (parameter identifier: MQCACF_GROUP_ENTITY_NAMES).

The names of groups having their authorizations set. At least one group name or principal name must be specified. An error occurs if neither are specified.

Each member in this list can be a maximum length of MQ_ENTITY_NAME_LENGTH.

PrincipalNames **(MQCFSL)**
Principal names (parameter identifier: MQCACF_PRINCIPAL_ENTITY_NAMES).

The names of principals having their authorizations set. At least one group name or principal name must be specified. An error occurs if neither are specified.

Each member in this list can be a maximum length of MQ_ENTITY_NAME_LENGTH.

ServiceComponent **(MQCFST)**
Service component (parameter identifier: MQCACF_SERVICE_COMPONENT).

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply.

If you omit this parameter, the authorization inquiry is made to the first installable component for the service.

The maximum length of the string is MQ_SERVICE_COMPONENT_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRC_UNKNOWN_ENTITY
Userid not authorized, or unknown.

MQRCCF_AUTH_VALUE_ERROR
Invalid authorization.

MQRCCF_AUTH_VALUE_MISSING
Authorization missing.

MQRCCF_ENTITY_NAME_MISSING
Entity name missing.

MQRCCF_OBJECT_TYPE_MISSING
Object type missing.

MQRCCF_PROFILE_NAME_ERROR
Invalid profile name.

Set Channel Authentication Record:

The Set Channel Authentication Record (MQCMD_SET_CHLAUTH_REC) command sets the allowed partner details and mappings to MCAUSER for a channel or set of channels.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Syntax diagram

See the syntax diagram in the MQSC "SET CHLAUTH" on page 772 command for combinations of parameters and values that are allowed.

Required parameters

The required parameters are valid for the **Action** values of:
- MQACT_ADD or MQACT_REPLACE
- MQACT_REMOVE
- MQACT_REMOVEALL

ProfileName **(MQCFST)**
The name of the channel or set of channels for which you are setting channel authentication configuration (parameter identifier: MQCACH_CHANNEL_NAME). You can use one or more asterisks (*), in any position, as wildcards to specify a set of channels. If you set Type to MQCAUT_BLOCKADDR, you must set the generic channel name to a single asterisk, which matches all channel names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Type **(MQCFIN)**
The **Type** parameter must follow the **ProfileName** parameter.

The type of channel authentication record for which to set allowed partner details or mappings to MCAUSER (parameter identifier: MQIACF_CHLAUTH_TYPE). The following values are valid:

MQCAUT_BLOCKUSER
This channel authentication record prevents a specified user or users from connecting. The MQCAUT_BLOCKUSER parameter must be accompanied by a UserList.

MQCAUT_BLOCKADDR
This channel authentication record prevents connections from a specified IP address or addresses. The MQCAUT_BLOCKADDR parameter must be accompanied by an AddrList.

MQCAUT_SSLPEERMAP
This channel authentication record maps SSL Distinguished Names (DNs) to MCAUSER values. The MQCAUT_SSLPEERMAP parameter must be accompanied by an SSLPeer.

MQCAUT_ADDRESSMAP
This channel authentication record maps IP addresses to MCAUSER values. The MQCAUT_ADDRESSMAP parameter must be accompanied by an Address.

MQCAUT_USERMAP
This channel authentication record maps asserted user IDs to MCAUSER values. The MQCAUT_USERMAP parameter must be accompanied by a ClntUser.

MQCAUT_QMGRMAP
This channel authentication record maps remote queue manager names to MCAUSER values. The MQCAUT_QMGRMAP parameter must be accompanied by a QMName.

Optional parameters

The following table shows which parameters are valid for each value of **Action**:

	Action		
Parameter	**MQACT_ADD or MQACT_REPLACE**	**MQACT_REMOVE**	**MQACT_REMOVEALL**
CommandScope	✓	✓	✓
Action	✓	✓	✓
Address	✓	✓	
Addrlist	✓	✓	
ClntUser	✓	✓	
MCAUser	✓		
QMName	✓	✓	
SSLPeer	✓	✓	
UserList	✓	✓	
UserSrc	✓		
Warn	✓		
Description	✓		

Action **(MQCFIN)**
The action to perform on the channel authentication record (parameter identifier: MQIACF_ACTION). The following values are valid:

MQACT_ADD
Add the specified configuration to a channel authentication record. This is the default value.

For types MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP and MQCAUT_QMGRMAP, if the specified configuration exists, the command fails.

For types MQCAUT_BLOCKUSER and MQCAUT_BLOCKADDR, the configuration is added to the list.

MQACT_REPLACE
Replace the current configuration of a channel authentication record.

For types MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP and MQCAUT_QMGRMAP, if the specified configuration exists, it is replaced with the new configuration. If it does not exist it is added.

For types MQCAUT_BLOCKUSER and MQCAUT_BLOCKADDR, the configuration specified replaces the current list, even if the current list is empty. If you replace the current list with an empty list, this acts like MQACT_REMOVEALL.

MQACT_REMOVE
Remove the specified configuration from the channel authentication records. If the configuration does not exist the command fails. If you remove the last entry from a list, this acts like MQACT_REMOVEALL.

MQACT_REMOVEALL
Remove all members of the list and thus the whole record (for MQCAUT_BLOCKADDR and MQCAUT_BLOCKUSER) or all previously defined mappings (for MQCAUT_ADDRESSMAP, MQCAUT_SSLPEERMAP, MQCAUT_QMGRMAP and MQCAUT_USERMAP) from the channel authentication records. This option cannot be combined with specific values supplied in **AddrList**, **UserList**, **Address**, **SSLPeer**, **QMName** or **ClntUser**. If the specified type has no current configuration the command still succeeds.

Address **(MQCFST)**
The filter to be used to compare with the IP address of the partner queue manager or client at the other end of the channel (parameter identifier: MQCACH_CONNECTION_NAME).

This parameter is mandatory when **Type** is MQCAUT_ADDESSMAP and is also valid when **Type** is MQCAUT_SSLPEERMAP, MQCAUT_USERMAP, or MQCAUT_QMGRMAP and **Action** is MQACT_ADD, MQACT_REPLACE, or MQACT_REMOVE. You can define more than one channel authentication object with the same main identity, for example the same SSL or TLS peer name, with different addresses. See "Generic IP addresses" on page 778 for more information about filtering IP addresses.

The maximum length of the string is MQ_CONN_NAME_LENGTH.

AddrList **(MQCFSL)**
A list of up to 100 generic IP addresses which are banned from accessing this queue manager on any channel (parameter identifier: MQCACH_CONNECTION_NAME_LIST).

This parameter is only valid when **Type** is MQCAUT_BLOCKADDR.

The maximum length of each address is MQ_CONN_NAME_LENGTH.

ClntUser **(MQCFST)**
The client asserted user ID to be mapped to a new user ID or blocked (parameter identifier: MQCACH_CLIENT_USER_ID).

This parameter is valid only when **Type** is MQCAUT_BLOCKADDR.

The maximum length of the string is MQ_MCA_USER_ID_LENGTH.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is run when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is run on the queue manager on which it was entered.
- a queue manager name. The command is run on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which the command was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

- an asterisk (*). The command is run on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

Custom **(MQCFST)**
Reserved for future use.

Description **(MQCFST)**
Provides descriptive information about the channel authentication record, which is displayed when you issue the Inquire Channel Authentication Records command (parameter identifier: MQCA_CHLAUTH_DESC).

This parameter must contain only displayable characters. In a DBCS installation, it can contain DBCS characters. The maximum length of the string is MQ_CHLAUTH_DESC_LENGTH.

Note: Use characters from the coded character set identifier (CCSID) for this queue manager. Other characters might be translated incorrectly if the information is sent to another queue manager.

MCAUser **(MQCFST)**
The user identifier to be used when the inbound connection matches the SSL DN, IP address, client asserted user ID or remote queue manager name supplied (parameter identifier: MQCACH_MCA_USER_ID).

This parameter is mandatory when **UserSrc** is MQUSRC_MAP and is valid when **Type** is MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP, or MQCAUT_QMGRMAP.

This parameter is valid only when **Action** is MQACT_ADD or MQACT_REPLACE.

The maximum length of the string is MQ_MCA_USER_ID_LENGTH.

QMName **(MQCFST)**
The name of the remote partner queue manager, or pattern that matches a set of queue manager names, to be mapped to a user ID or blocked (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

This parameter is valid only when **Type** is MQCAUT_QMGRMAP

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

SSLPeer **(MQCFST)**

The filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel (parameter identifier: MQCACH_SSL_PEER_NAME).

The **SSLPeer** value is specified in the standard form used to specify a Distinguished Name. See Distinguished Names and WebSphere MQ rules for SSLPEER values.

The maximum length of the string is MQ_SSL_PEER_NAME_LENGTH .

UserList **(MQCFSL)**
A list of up to 100 user IDs which are banned from using this channel or set of channels (parameter identifier: MQCACH_MCA_USER_ID_LIST).

The following special value can be used:

***MQADMIN**
The exact meaning of this value is determined at runtime. If you are using the OAM supplied with IBM WebSphere MQ, the meaning depends on platform, as follows:

- On Windows, all members of the mqm group, the Administrators group and SYSTEM
- On UNIX and Linux, all members of the mqm group
- On IBM i, the profiles (users) qmqm and qmqmadm and all members of the qmqmadm group, and any user defined with the *ALLOBJ special setting

- On z/OS, the user ID that the CHINIT and the user ID that the MSTR address spaces are running under

This parameter is only valid when **TYPE** is MQCAUT_BLOCKUSER.

The maximum length of each user ID is MQ_MCA_USER_ID_LENGTH .

UserSrc **(MQCFIN)**
The source of the user ID to be used for MCAUSER at run time (parameter identifier: MQIACH_USER_SOURCE).

The following values are valid:

MQUSRC_MAP
Inbound connections that match this mapping use the user ID specified in the **MCAUser** attribute. This is the default value.

MQUSRC_NOACCESS
Inbound connections that match this mapping have no access to the queue manager and the channel ends immediately.

MQUSRC_CHANNEL
Inbound connections that match this mapping use the flowed user ID or any user defined on the channel object in the MCAUSER field.

Note that *Warn* and MQUSRC_CHANNEL, or MQUSRC_MAP are incompatible. This is because channel access is never blocked in these cases, so there is never a reason to generate a warning.

Warn **(MQCFIN)**
Indicates whether this record operates in warning mode (parameter identifier: MQIACH_WARNING).

MQWARN_NO
This record does not operate in warning mode. Any inbound connection that matches this record is blocked. This is the default value.

MQWARN_YES
This record operates in warning mode. Any inbound connection that matches this record and would therefore be blocked is allowed access. An error message is written and, if events are configured, an event message is created showing the details of what would have been blocked. The connection is allowed to continue. An attempt is made to find another record that is set to WARN(NO) to set the credentials for the inbound channel.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown at "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHLAUTH_TYPE_ERROR
Channel authentication record type not valid.

MQRCCF_CHLAUTH_ACTION_ERROR
Channel authentication record action not valid.

MQRCCF_CHLAUTH_USERSRC_ERROR
Channel authentication record user source not valid.

MQRCCF_WRONG_CHLAUTH_TYPE
Parameter not allowed for this channel authentication record type.

MQRCCF_CHLAUTH_ALREADY_EXISTS
Channel authentication record already exists

Related information:

Channel authentication records

Start Channel:

The Start Channel (MQCMD_START_CHANNEL) command starts a IBM WebSphere MQ channel. This command can be issued to a channel of any type (except MQCHT_CLNTCONN). If, however, it is issued to a channel with a *ChannelType* value of MQCHT_RECEIVER, MQCHT_SVRCONN, or MQCHT_CLUSRCVR, the only action is to enable the channel, not start it.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

None of the following attributes are applicable to MQTT channels unless specifically mentioned in the parameter description.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be started. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required for all channel types including MQTT channels.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ChannelDisposition **(MQCFIN)**
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be started.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

MQCHLD_FIXSHARED
Shared channels tied to a specific queue manager.

The combination of the *ChannelDisposition* and *CommandScope* parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On every active queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 104

Table 104. ChannelDisposition and CommandScope for START CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_PRIVATE	Start as a private channel on the local queue manager	Start as a private channel on the named queue manager	Start as a private channel on all active queue managers

Table 104. ChannelDisposition and CommandScope for START CHANNEL (continued)

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_SHARED	For channels of *ChannelType* MQCHT_SENDER, MQCHT_REQUESTER, and MQCHT_SERVER, start as a shared channel on the most suitable queue manager in the group. For a shared channel of *ChannelType* MQCHT_RECEIVER and MQCHT_SVRCONN, start the channel on all active queue managers. For a shared channel of *ChannelType* MQCHT_CLUSSDR and MQCHT_CLUSRCVR, this option is not permitted. MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted
MQCHLD_FIXSHARED	For a shared channel of *ChannelType* MQCHT_SENDER, MQCHT_REQUESTER, and MQCHT_SERVER, with a nonblank *ConnectionName*, start as a shared channel on the local queue manager.	For a shared channel of *ChannelType* MQCHT_SENDER, MQCHT_REQUESTER, and MQCHT_SERVER, with a nonblank *ConnectionName*, start as a shared channel on the named queue manager.	Not permitted

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_INDOUBT
Channel in-doubt.

MQRCCF_CHANNEL_IN_USE
Channel in use.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQINQ_FAILED
MQINQ call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_NOT_XMIT_Q
Queue is not a transmission queue.

Start Channel (MQTT):

The Start Channel (MQCMD_START_CHANNEL) command starts a IBM WebSphere MQ channel. This command can be issued to a channel of type MQCHT_MQTT.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be started. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required for all channel types including MQTT channels.

ChannelType **(MQCFIN)**
The type of channel (parameter identifier: MQIACH_CHANNEL_TYPE). This parameter is currently only used with MQTT Telemetry channels, and is required when starting a Telemetry channel. The only value that can currently be given to the parameter is MQCHT_MQTT.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_PARM_SYNTAX_ERROR
The parameter specified contained a syntax error.

MQRCCF_PARM_MISSING
Parameters are missing.

MQRCCF_CHANNEL_NOT_FOUND
The channel specified does not exist.

MQRCCF_CHANNEL_IN_USE
The command did not specify a parameter or parameter value that was required.

MQRCCF_NO_STORAGE
Insufficient storage is available.

MQRCCF_COMMAND_FAILED
The command has failed.

MQRCCF_PORT_IN_USE
The port is in use.

MQRCCF_BIND_FAILED
The bind to a remote system during session negotiation has failed.

MQRCCF_SOCKET_ERROR
Socket error has occurred.

MQRCCF_HOST_NOT_AVAILABLE
An attempt to allocate a conversation to a remote system was unsuccessful. The error might be transitory, and the allocate might succeed later. This reason can occur if the listening program at the remote system is not running.

Start Channel Initiator:

The Start Channel Initiator (MQCMD_START_CHANNEL_INIT) command starts a WebSphere MQ channel initiator.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

InitiationQName **(MQCFST)**
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The name of the initiation queue for the channel initiation process. That is, the initiation queue that is specified in the definition of the transmission queue.

This parameter is not valid on z/OS.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

EnvironmentInfo **(MQCFST)**
Environment information (parameter identifier: MQCACF_ENV_INFO).

The parameters and values to be substituted in the JCL procedure (xxxxCHIN, where xxxx is the queue manager name) that is used to start the channel initiator address space. This parameter applies to z/OS only.

The maximum length of the string is MQ_ENV_INFO_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQGET_FAILED
MQGET call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

Start Channel Listener:

The Start Channel Listener (MQCMD_START_CHANNEL_LISTENER) command starts a WebSphere MQ listener. On z/OS, this command is valid for any transmission protocol; on other platforms, it is valid only for TCP transmission protocols.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_Q_MGR_NAME_LENGTH.

InboundDisposition **(MQCFIN)**
Inbound transmission disposition (parameter identifier: MQIACH_INBOUND_DISP). This parameter applies to z/OS only.

Specifies the disposition of the inbound transmissions that are to be handled. The value can be:

MQINBD_Q_MGR
Listen for transmissions directed to the queue manager. MQINBD_Q_MGR is the default.

MQINBD_GROUP
Listen for transmissions directed to the queue-sharing group. MQINBD_GROUP is permitted only if there is a shared queue manager environment.

IPAddress **(MQCFST)**
IP address (parameter identifier: MQCACH_IP_ADDRESS). This parameter applies to z/OS only.

The IP address for TCP/IP specified in IPv4 dotted decimal, IPv6 hexadecimal, or alphanumeric form. This parameter is valid only for channels that have a *TransportType* of MQXPT_TCP.

The maximum length of the string is MQ_IP_ADDRESS_LENGTH.

ListenerName **(MQCFST)**
Listener name (parameter identifier: MQCACH_LISTENER_NAME). This parameter does not apply to z/OS.

The name of the listener definition to be started. On those platforms on which this parameter is valid, if this parameter is not specified, the default listener SYSTEM.DEFAULT.LISTENER is assumed. If this parameter is specified, no other parameters can be specified.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

LUName **(MQCFST)**
LU name (parameter identifier: MQCACH_LU_NAME). This parameter applies to z/OS only.

The symbolic destination name for the logical unit (LU) as specified in the APPC side information data set. The LU must be the same LU that is specified in the channel initiator parameters to be used for outbound transmissions. This parameter is valid only for channels with a *TransportType* of MQXPT_LU62.

The maximum length of the string is MQ_LU_NAME_LENGTH.

Port **(MQCFIN)**
Port number for TCP (parameter identifier: MQIACH_PORT_NUMBER). This parameter applies to z/OS only.

The port number for TCP. This parameter is valid only for channels with a *TransportType* of MQXPT_TCP.

TransportType **(MQCFIN)**
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

On platforms other than z/OS, this parameter is invalid.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_COMMS_LIBRARY_ERROR
Communications protocol library error.

MQRCCF_LISTENER_NOT_STARTED
Listener not started.

MQRCCF_LISTENER_RUNNING
Listener already running.

MQRCCF_NETBIOS_NAME_ERROR
NetBIOS listener name error.

Start Service:

The Start Service (MQCMD_START_SERVICE) command starts an existing WebSphere MQ service definition.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ServiceName **(MQCFST)**
Service name (parameter identifier: MQCA_SERVICE_NAME).

This parameter is the name of the service definition to be started. The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in “Error codes applicable to all commands” on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_NO_START_CMD
The *StartCommand* parameter of the service is blank.

MQRCCF_SERVICE_RUNNING
Service is already running.

Stop Channel:

The Stop Channel (MQCMD_STOP_CHANNEL) command stops a IBM WebSphere MQ channel.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

This command can be issued to a channel of any type (except MQCHT_CLNTCONN).

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

None of the following attributes are applicable to MQTT channels unless specifically mentioned in the parameter description.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be stopped. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required for all channel types..

Optional parameters

ChannelDisposition **(MQCFIN)**
Channel disposition (parameter identifier: MQIACH_CHANNEL_DISP). This parameter applies to z/OS only.

Specifies the disposition of the channels to be stopped.

If this parameter is omitted, then the value for the channel disposition is taken from the default channel disposition attribute of the channel object.

The value can be:

MQCHLD_PRIVATE
A receiving channel is private if it was started in response to an inbound transmission directed to the queue manager.

A sending channel is private if its transmission queue has a disposition other than MQQSGD_SHARED.

MQCHLD_SHARED
A receiving channel is shared if it was started in response to an inbound transmission directed to the queue-sharing group.

A sending channel is shared if its transmission queue has a disposition of MQQSGD_SHARED.

The combination of the *ChannelDisposition* and *CommandScope* parameters also controls from which queue manager the channel is operated. The possible options are:

- On the local queue manager where the command is issued.
- On another specific named queue manager in the group.
- On every active queue manager in the group.
- On the most suitable queue manager in the group, determined automatically by the queue manager itself.

The various combinations of *ChannelDisposition* and *CommandScope* are summarized in Table 105

Table 105. ChannelDisposition and CommandScope for STOP CHANNEL

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_PRIVATE	Stop as a private channel on the local queue manager	Stop as a private channel on the named queue manager	Stop as a private channel on all active queue managers

Table 105. ChannelDisposition and CommandScope for STOP CHANNEL (continued)

ChannelDisposition	*CommandScope* **blank or local-qmgr**	*CommandScope* **qmgr-name**	*CommandScope(*)*
MQCHLD_SHARED	For channels of *ChannelType* MQCHT_RECEIVER or MQCHT_SVRCONN, stop as shared channel on all active queue managers. For channels of *ChannelType* MQCHT_SENDER, MQCHT_REQUESTER, and MQCHT_SERVER, stop as a shared channel on the queue manager where it is running. If the channel is in an inactive state (not running), or if it is in RETRY state because the channel initiator on which it was running has stopped, a STOP request for the channel is issued on the local queue manager. MQCHLD_SHARED might automatically generate a command using *CommandScope* and send it to the appropriate queue manager. If there is no definition for the channel on the queue manager to which the command is sent, or if the definition is unsuitable for the command, the command fails. The definition of a channel on the queue manager where the command is entered might be used to determine the target queue manager where the command is run. Therefore, it is important that channel definitions are consistent. Inconsistent channel definitions might result in unexpected command behavior.	Not permitted	Not permitted

ChannelStatus **(MQCFIN)**
The new state of the channel after the command is executed (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be:

MQCHS_INACTIVE
Channel is inactive.

MQCHS_STOPPED
Channel is stopped. MQCHS_STOPPED is the default if nothing is specified.

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.
- an asterisk (*). The command is executed on the local queue manager and is also passed to every active queue manager in the queue-sharing group.

The maximum length is MQ_QSG_NAME_LENGTH.

ConnectionName **(MQCFST)**
Connection name of channel to be stopped (parameter identifier: MQCACH_CONNECTION_NAME).

This parameter is the connection name of the channel to be stopped. If this parameter is omitted, all channels with the specified channel name and remote queue manager name are stopped. On platforms other than z/OS, the maximum length of the string is MQ_CONN_NAME_LENGTH. On z/OS, the maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

If this parameter is specified, ChannelStatus must be MQCHS_INACTIVE.

Mode **(MQCFIN)**
How the channel must be stopped (parameter identifier: MQIACF_MODE).

The value can be:

MQMODE_QUIESCE
Quiesce the channel. MQMODE_QUIESCE is the default.

If you issue a `Stop Channel <channelname> Mode(MQMODE_QUIESCE)` command on a server-connection channel with the sharing conversations feature enabled, the IBM WebSphere MQ client infrastructure becomes aware of the stop request in a timely manner; this time is dependent upon the speed of the network. The client application becomes aware of the stop request as a result of issuing a subsequent call to IBM WebSphere MQ.

MQMODE_FORCE
Stop the channel immediately; the thread or process of the channel is not terminated. Stops transmission of any current batch.

For server-connection channels, breaks the current connection, returning MQRC_CONNECTION_BROKEN.

For other types of channels, this situation is likely to result in in-doubt situations.

Onz/OS, this option interrupts any message reallocation in progress, which can leave BIND_NOT_FIXED messages partially reallocated or out of order.

MQMODE_TERMINATE
On z/OS, MQMODE_TERMINATE is synonymous with FORCE. On other platforms, stop the channel immediately; the thread or process of the channel is terminated.

On z/OS, this option interrupts any message reallocation in progress, which can leave BIND_NOT_FIXED messages partially reallocated or out of order.

Note: This parameter was previously called *Quiesce* (MQIACF_QUIESCE), with values MQQO_YES and MQQO_NO. The old names can still be used.

QMgrName **(MQCFST)**
Name of remote queue manager (parameter identifier: MQCA_Q_MGR_NAME).

This parameter is the name of the remote queue manager to which the channel is connected. If this parameter is omitted, all channels with the specified channel name and connection name are stopped. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

If this parameter is specified, ChannelStatus must be MQCHS_INACTIVE.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_DISABLED
Channel disabled.

MQRCCF_CHANNEL_NOT_ACTIVE
Channel not active.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_MODE_VALUE_ERROR
Mode value not valid.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_MQSET_FAILED
MQSET call failed.

Stop Channel (MQTT):

The Stop Channel (MQCMD_STOP_CHANNEL) command stops a IBM WebSphere MQ Telemetry channel.

Required parameters

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

This parameter is required.

The name of the channel to be stopped. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelType **(MQCFIN)**
The type of channel (parameter identifier: MQIACH_CHANNEL_TYPE). This parameter is currently only used with MQTT Telemetry channels, and is required when stopping a Telemetry channel. The only value that can currently be given to the parameter is **MQCHT_MQTT**.

ClientIdentifier **(MQCFST)**
Client identifier. The client identifier is a 23-byte string that identifies a IBM WebSphere MQ Telemetry Transport client. When the Stop Channel command specifies a *ClientIdentifier*, only the connection for the specified client identifier is stopped. If the CLIENTID is not specified, all the connections on the channel are stopped.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CHANNEL_DISABLED
Channel disabled.

MQRCCF_CHANNEL_NOT_ACTIVE
Channel not active.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_MODE_VALUE_ERROR
Mode value not valid.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_MQSET_FAILED
MQSET call failed.

Stop Channel Listener:

The Stop Channel Listener (MQCMD_STOP_CHANNEL_LISTENER) command stops a WebSphere MQ listener.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ListenerName **(MQCFST)**
Listener name (parameter identifier: MQCACH_LISTENER_NAME). This parameter does not apply to z/OS.

The name of the listener definition to be stopped. If this parameter is specified, no other parameters can be specified.

The maximum length of the string is MQ_LISTENER_NAME_LENGTH.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE).

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

This parameter is valid only on z/OS.

The maximum length is MQ_QSG_NAME_LENGTH.

InboundDisposition **(MQCFIN)**
Inbound transmission disposition (parameter identifier: MQIACH_INBOUND_DISP).

Specifies the disposition of the inbound transmissions that the listener handles. The value can be:

MQINBD_Q_MGR
Handling for transmissions directed to the queue manager. MQINBD_Q_MGR is the default.

MQINBD_GROUP
Handling for transmissions directed to the queue-sharing group. MQINBD_GROUP is permitted only if there is a shared queue manager environment.

This parameter is valid only on z/OS.

IPAddress **(MQCFST)**
IP address (parameter identifier: MQCACH_IP_ADDRESS).

The IP address for TCP/IP specified in dotted decimal or alphanumeric form. This parameter is valid on z/OS only where channels have a *TransportType* of MQXPT_TCP.

The maximum length of the string is MQ_IP_ADDRESS_LENGTH.

This parameter is valid only on z/OS.

Port **(MQCFIN)**
Port number for TCP (parameter identifier: MQIACH_PORT_NUMBER).

The port number for TCP. This parameter is valid only on z/OS where channels have a *TransportType* of MQXPT_TCP.

TransportType **(MQCFIN)**
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

This parameter is valid only on z/OS.

Error codes

This command might return the following error code in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_LISTENER_STOPPED
Listener not running.

Stop Connection:

The Stop Connection (MQCMD_STOP_CONNECTION) command attempts to break a connection between an application and the queue manager. There might be circumstances in which the queue manager cannot implement this command.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ConnectionId **(MQCFBS)**
Connection identifier (parameter identifier: MQBACF_CONNECTION_ID).

This parameter is the unique connection identifier associated with an application that is connected to the queue manager.

The length of the byte string is MQ_CONNECTION_ID_LENGTH.

Stop Service:

The Stop Service (MQCMD_STOP_SERVICE) command stops an existing WebSphere MQ service definition that is running.

HP Integrity NonStop Server	UNIX and Linux	Windows
	X	X

Required parameters

ServiceName **(MQCFST)**
Service name (parameter identifier: MQCA_SERVICE_NAME).

This parameter is the name of the service definition to be stopped. The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown on page "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_NO_STOP_CMD
The *StopCommand* parameter of the service is blank.

MQRCCF_SERVICE_STOPPED
Service is not running.

Suspend Queue Manager Cluster:

The Suspend Queue Manager Cluster (MQCMD_SUSPEND_Q_MGR_CLUSTER) command informs other queue managers in a cluster that the local queue manager is not available for processing, and cannot be sent messages. Its action can be reversed by the Resume Queue Manager Cluster (MQCMD_RESUME_Q_MGR_CLUSTER) command.

HP Integrity NonStop Server	UNIX and Linux	Windows
X	X	X

Required parameters

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster for which availability is to be suspended.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist **(MQCFST)**
Cluster Namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist specifying a list of clusters for which availability is to be suspended.

Optional parameters

CommandScope **(MQCFST)**
Command scope (parameter identifier: MQCACF_COMMAND_SCOPE). This parameter applies to z/OS only.

Specifies how the command is executed when the queue manager is a member of a queue-sharing group. You can specify one of the following:

- blank (or omit the parameter altogether). The command is executed on the queue manager on which it was entered.
- a queue manager name. The command is executed on the queue manager you specify, providing it is active within the queue sharing group. If you specify a queue manager name other than the queue manager on which it was entered, you must be using a queue-sharing group environment, and the command server must be enabled.

The maximum length is MQ_QSG_NAME_LENGTH.

Mode **(MQCFIN)**
How the local queue manager is suspended from the cluster (parameter identifier: MQIACF_MODE).

The value can be:

MQMODE_QUIESCE
Other queue managers in the cluster are told not to send further messages to the local queue manager.

MQMODE_FORCE
All inbound and outbound channels to other queue managers in the cluster are stopped forcibly.

Note: This parameter was previously called *Quiesce* (MQIACF_QUIESCE), with values MQQO_YES and MQQO_NO. The old names can still be used.

Error codes

This command might return the following error codes in the response format header, in addition to the values shown in "Error codes applicable to all commands" on page 802.

Reason **(MQLONG)**
The value can be:

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_MODE_VALUE_ERROR
Mode value not valid.

Structures for commands and responses

PCF commands and responses have a consistent structure including of a header and any number of parameter structures of defined types.

Commands and responses have the form:

- PCF header (MQCFH) structure (described in topic "MQCFH - PCF header" on page 1215), followed by
- Zero or more parameter structures. Each of these is one of the following:
  - PCF byte string filter parameter (MQCFBF, see topic "MQCFBF - PCF byte string filter parameter" on page 1219)
  - PCF byte string parameter (MQCFBS, see topic "MQCFBS - PCF byte string parameter" on page 1222)
  - PCF integer filter parameter (MQCFIF, see topic "MQCFIF - PCF integer filter parameter" on page 1224)
  - PCF integer list parameter (MQCFIL, see topic "MQCFIL - PCF integer list parameter" on page 1227)
  - PCF integer parameter (MQCFIN, see topic "MQCFIN - PCF integer parameter" on page 1229)
  - PCF string filter parameter (MQCFSF, see topic "MQCFSF - PCF string filter parameter" on page 1231)
  - PCF string list parameter (MQCFSL, see topic "MQCFSL - PCF string list parameter" on page 1235)
  - PCF string parameter (MQCFST, see topic "MQCFST - PCF string parameter" on page 1238)

How the structures are shown:

The structures are described in a language-independent form.

The declarations are shown in the following programming languages:

- C
- COBOL
- PL/I
- S/390® assembler
- Visual Basic

Data types

For each field of the structure, the data type is given in brackets after the field name. These data types are the elementary data types described in Data types used in the MQI.

Initial values and default structures

See WebSphere MQ COPY, header, include, and module files for details of the supplied header files that contain the structures, constants, initial values, and default structures.

Usage notes:

The format of the strings in the PCF message determines the settings of the character set fields in the message descriptor to enable conversion of strings within the message.

If all of the strings in a PCF message have the same coded character-set identifier, the *CodedCharSetId* field in the message descriptor MQMD should be set to that identifier when the message is put, and the *CodedCharSetId* fields in the MQCFST, MQCFSL, and MQCFSF structures within the message should be set to MQCCSI_DEFAULT.

If the format of the PCF message is MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF and some of the strings in the message have different character-set identifiers, the *CodedCharSetId* field in MQMD should be set to MQCCSI_EMBEDDED when the message is put, and the *CodedCharSetId* fields in the MQCFST, MQCFSL, and MQCFSF structures within the message should all be set to the identifiers that apply.

This enables conversions of the strings within the message, to the *CodedCharSetId* value in the MQMD specified on the MQGET call, if the MQGMO_CONVERT option is also specified.

For more information about the MQEPH structure, see MQEPH - Embedded PCF header.

Note: If you request conversion of the internal strings in the message, the conversion will occur only if the value of the *CodedCharSetId* field in the MQMD of the message is different from the *CodedCharSetId* field of the MQMD specified on the MQGET call.

Do not specify MQCCSI_EMBEDDED in MQMD when the message is put, with MQCCSI_DEFAULT in the MQCFST, MQCFSL, or MQCFSF structures within the message, as this will prevent conversion of the message.

MQCFH - PCF header:

The MQCFH structure describes the information that is present at the start of the message data of a command message, or a response to a command message. In either case, the message descriptor *Format* field is MQFMT_ADMIN.

The PCF structures are also used for event messages. In this case the message descriptor *Format* field is MQFMT_EVENT.

The PCF structures can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength* and *ParameterCount* fields to the values appropriate to the data.

Fields for MQCFH

Type **(MQLONG)**
Structure type.

This field indicates the content of the message. The following are valid for commands:

MQCFT_COMMAND
Message is a command.

MQCFT_COMMAND_XR
Message is a command to which standard or extended responses might be sent.

This value is required on z/OS.

MQCFT_RESPONSE
Message is a response to a command.

MQCFT_XR_MSG
Message is an extended response to a command. It contains informational or error details.

MQCFT_XR_ITEM
Message is an extended response to an Inquire command. It contains item data.

MQCFT_XR_SUMMARY
Message is an extended response to a command. It contains summary information.

MQCFT_USER
User-defined PCF message.

StrucLength **(MQLONG)**
Structure length.

This field is the length in bytes of the MQCFH structure. The value must be:

MQCFH_STRUC_LENGTH
Length of command format header structure.

Version **(MQLONG)**
Structure version number.

For z/OS, the value must be:

MQCFH_VERSION_3
Version number for command format header structure.

The following constant specifies the version number of the current version:

MQCFH_CURRENT_VERSION
Current version of command format header structure.

Command **(MQLONG)**
Command identifier.

For a command message, this field identifies the function to be performed. For a response message, it identifies the command to which this field is the reply. See the description of each command for the value of this field.

MsgSeqNumber **(MQLONG)**
Message sequence number.

This field is the sequence number of the message within a set of related messages. For a command, this field must have the value one (because a command is always contained within a single message). For a response, the field has the value one for the first (or only) response to a command, and increases by one for each successive response to that command.

The last (or only) message in a set has the MQCFC_LAST flag set in the *Control* field.

Control **(MQLONG)**
Control options.

The following are valid:

MQCFC_LAST
Last message in the set.

For a command, this value must always be set.

MQCFC_NOT_LAST
Not the last message in the set.

CompCode **(MQLONG)**
Completion code.

This field is meaningful only for a response; its value is not significant for a command. The following are possible:

MQCC_OK
Command completed successfully.

MQCC_WARNING
Command completed with warning.

MQCC_FAILED
Command failed.

MQCC_UNKNOWN
Whether command succeeded is not known.

Reason **(MQLONG)**
Reason code qualifying completion code.

This field is meaningful only for a response; its value is not significant for a command.

The possible reason codes that can be returned in response to a command are listed in,"Definitions of the Programmable Command Formats" on page 796 and in the description of each command.

ParameterCount **(MQLONG)**
Count of parameter structures.

This field is the number of parameter structures (MQCFBF, MQCFBS, MQCFIF, MQCFIL, MQCFIN, MQCFSL, MQCFSF, and MQCFST) that follow the MQCFH structure. The value of this field is zero or greater.

C language declaration

```
typedef struct tagMQCFH {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Version;         /* Structure version number */
  MQLONG  Command;         /* Command identifier */
  MQLONG  MsgSeqNumber;    /* Message sequence number */
  MQLONG  Control;         /* Control options */
  MQLONG  CompCode;        /* Completion code */
  MQLONG  Reason;          /* Reason code qualifying completion code */
  MQLONG  ParameterCount;  /* Count of parameter structures */
 } MQCFH;
```

COBOL language declaration

```
**   MQCFH structure
  10 MQCFH.
**    Structure type
   15 MQCFH-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFH-STRUCLENGTH    PIC S9(9) BINARY.
**    Structure version number
   15 MQCFH-VERSION        PIC S9(9) BINARY.
**    Command identifier
   15 MQCFH-COMMAND        PIC S9(9) BINARY.
**    Message sequence number
   15 MQCFH-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Control options
   15 MQCFH-CONTROL        PIC S9(9) BINARY.
**    Completion code
```

```
   15 MQCFH-COMPCODE       PIC S9(9) BINARY.
**    Reason code qualifying completion code
   15 MQCFH-REASON         PIC S9(9) BINARY.
**    Count of parameter structures
   15 MQCFH-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFH based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Version        fixed bin(31), /* Structure version number */
  3 Command        fixed bin(31), /* Command identifier */
  3 MsgSeqNumber   fixed bin(31), /* Message sequence number */
  3 Control        fixed bin(31), /* Control options */
  3 CompCode       fixed bin(31), /* Completion code */
  3 Reason         fixed bin(31), /* Reason code qualifying completion
                                     code */
  3 ParameterCount fixed bin(31); /* Count of parameter structures */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFH                           DSECT
MQCFH_TYPE                      DS   F        Structure type
MQCFH_STRUCLENGTH               DS   F        Structure length
MQCFH_VERSION                   DS   F        Structure version number
MQCFH_COMMAND                   DS   F        Command identifier
MQCFH_MSGSEQNUMBER              DS   F        Message sequence number
MQCFH_CONTROL                   DS   F        Control options
MQCFH_COMPCODE                  DS   F        Completion code
MQCFH_REASON                    DS   F        Reason code qualifying
*                                             completion code
MQCFH_PARAMETERCOUNT            DS   F        Count of parameter
*                                             structures
MQCFH_LENGTH                    EQU  *-MQCFH  Length of structure
                                ORG  MQCFH
MQCFH_AREA                      DS   CL(MQCFH_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFH
  Type As Long            'Structure type
  StrucLength As Long     'Structure length
  Version As Long         'Structure version number
  Command As Long         'Command identifier
  MsgSeqNumber As Long    'Message sequence number
  Control As Long         'Control options
  CompCode As Long        'Completion code
  Reason As Long          'Reason code qualifying completion code
  ParameterCount As Long  'Count of parameter structures
End Type

Global MQCFH_DEFAULT As MQCFH
```

RPG language declaration (IBM i only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFH Structure
D*
D* Structure type
D  FHTYP                  1      4I 0 INZ(1)
D* Structure length
D  FHLEN                  5      8I 0 INZ(36)
D* Structure version number
D  FHVER                  9     12I 0 INZ(1)
D* Command identifier
D  FHCMD                 13     16I 0 INZ(0)
```

```
D* Message sequence number
D  FHSEQ                 17     20I 0 INZ(1)
D* Control options
D  FHCTL                 21     24I 0 INZ(1)
D* Completion code
D  FHCMP                 25     28I 0 INZ(0)
D* Reason code qualifying completion code
D  FHREA                 29     32I 0 INZ(0)
D* Count of parameter structures
D  FHCNT                 33     36I 0 INZ(0)
D*
```

MQCFBF - PCF byte string filter parameter:

The MQCFBF structure describes a byte string filter parameter. The format name in the message descriptor is MQFMT_ADMIN.

The MQCFBF structure is used in Inquire commands to provide a filter description. This filter description is used to filter the results of the Inquire command and return to the user only those objects that satisfy the filter description.

When an MQCFBF structure is present, the Version field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_3 or higher.

Fields for MQCFBF

Type **(MQLONG)**
Structure type.

This indicates that the structure is a MQCFBF structure describing a byte string filter parameter. The value must be:

MQCFT_BYTE_STRING_FILTER
Structure defining a byte string filter.

StrucLength **(MQLONG)**
Structure length.

This is the length, in bytes, of the MQCFBF structure, including the string at the end of the structure (the *FilterValue* field). The length must be a multiple of 4, and must be sufficient to contain the string. Bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *FilterValue* field:

MQCFBF_STRUC_LENGTH_FIXED
Length of fixed part of command format filter string-parameter structure.

Parameter **(MQLONG)**
Parameter identifier.

This identifies the parameter that is to be filtered on. The value of this identifier depends on the parameter to be filtered on.

The parameter is one of the following:

- MQBACF_EXTERNAL_UOW_ID
- MQBACF_Q_MGR_UOW_ID
- MQBACF_ORIGIN_UOW_ID (on z/OS only)

Operator **(MQLONG)**
Operator identifier.

This identifies the operator that is being used to evaluate whether the parameter satisfies the filter-value.

Possible values are:

MQCFOP_GREATER
Greater than

MQCFOP_LESS
Less than

MQCFOP_EQUAL
Equal to

MQCFOP_NOT_EQUAL
Not equal to

MQCFOP_NOT_LESS
Greater than or equal to

MQCFOP_NOT_GREATER
Less than or equal to

FilterValueLength **(MQLONG)**
Length of filter-value string.

This is the length, in bytes, of the data in the *FilterValue* field. This must be zero or greater, and does not need to be a multiple of 4.

FilterValue **(MQBYTE**×*FilterValueLength***)**
Filter value.

This specifies the filter-value that must be satisfied. Use this parameter where the response type of the filtered parameter is a byte string. Depending on the filter-keyword, this can be:

Note: If the specified byte string is shorter than the standard length of the parameter in MQFMT_ADMIN command messages, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.

C language declaration

```
typedef struct tagMQCFBF {
   MQLONG  Type;               /* Structure type */
   MQLONG  StrucLength;        /* Structure length */
   MQLONG  Parameter;          /* Parameter identifier */
   MQLONG  Operator;           /* Operator identifier */
   MQLONG  FilterValueLength;  /* Filter value length */
   MQBYTE  FilterValue[1];     /* Filter value -- first byte */
  } MQCFBF;
```

COBOL language declaration

```
**   MQCFBF structure
  10  MQCFBF.
** Structure type
  15  MQCFBF-TYPE PIC S9(9) BINARY.
** Structure length
  15  MQCFBF-STRUCLENGTH PIC S9(9) BINARY.
** Parameter identifier
  15  MQCFBF-PARAMETER PIC S9(9) BINARY.
** Operator identifier
  15  MQCFBF-OPERATOR PIC S9(9) BINARY.
** Filter value length
  15  MQCFBF-FILTERVALUELENGTH PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
  1 MQCFBF based,
   3 Type  fixed bin(31)
    init(MQCFT_BYTE_STRING_FILTER),  /* Structure type */
   3 StrucLength  fixed bin(31)
    init(MQCFBF_STRUC_LENGTH_FIXED),  /* Structure length */
   3 Parameter  fixed bin(31)
    init(0),  /* Parameter identifier */
   3 Operator  fixed bin(31)
    init(0),  /* Operator identifier */
   3 FilterValueLength  fixed bin(31)
    init(0);  /* Filter value length */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFBF                       DSECT
MQCFBF_TYPE                  DS  F    Structure type
MQCFBF_STRUCLENGTH           DS  F    Structure length
MQCFBF_PARAMETER             DS  F    Parameter identifier
MQCFBF_OPERATOR              DS  F    Operator identifier
MQCFBF_FILTERVALUELENGTH     DS  F    Filter value length
MQCFBF_LENGTH                EQU  *-MQCFIF Length of structure
                             ORG  MQCFBF
MQCFBF_AREA                  DS   CL(MQCFBF_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFBF
  Type As Long 'Structure type'
  StrucLength As Long 'Structure length'
  Parameter As Long 'Parameter identifier'
  Operator As Long 'Operator identifier'
  FilterValueLength As Long 'Filter value length'
  FilterValue As 1 'Filter value -- first byte'
End Type
Global MQCFBF_DEFAULT As MQCFBF
```

RPG language declaration (IBM i only)

```
D* MQCFBF Structure
D*
D* Structure type
D  FBFTYP                 1      4I 0 INZ(15)
D* Structure length
D  FBFLEN                 5      8I 0 INZ(20)
D* Parameter identifier
D  FBFPRM                 9     12I 0 INZ(0)
D* Operator identifier
D  FBFOP                 13     16I 0 INZ(0)
D* Filter value length
D  FBFFVL                17     20I 0 INZ(0)
D* Filter value -- first byte
D  FBFFV                 21     21    INZ
```

MQCFBS - PCF byte string parameter:

The MQCFBS structure describes a byte-string parameter in a PCF message. The format name in the message descriptor is MQFMT_ADMIN.

When an MQCFBS structure is present, the *Version* field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_2 or greater.

In a user PCF message, the *Parameter* field has no significance, and can be used by the application for its own purposes.

The structure ends with a variable-length byte string; see the *String* field in the following section for further details.

Fields for MQCFBS

Type **(MQLONG)**
Structure type.

This indicates that the structure is an MQCFBS structure describing byte string parameter. The value must be:

MQCFT_BYTE_STRING
Structure defining a byte string.

StrucLength **(MQLONG)**
Structure length.

This is the length in bytes of the MQCFBS structure, including the variable-length string at the end of the structure (the *String* field). The length must be a multiple of four, and must be sufficient to contain the string; any bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *String* field:

MQCFBS_STRUC_LENGTH_FIXED
Length of fixed part of MQCFBS structure.

Parameter **(MQLONG)**
Parameter identifier.

This identifies the parameter with a value that is contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1215 for details. In user PCF messages (MQCFT_USER), this field has no significance.

The parameter is from the MQBACF_* group of parameters.

StringLength **(MQLONG)**
Length of string.

This is the length in bytes of the data in the *string* field; it must be zero or greater. This length does not need to be a multiple of four.

String **(MQBYTE**×*StringLength***)**
String value.

This is the value of the parameter identified by the *parameter* field. The string is a byte string, and so is not subject to character-set conversion when sent between different systems.

Note: A null character in the string is treated as normal data, and does not act as a delimiter for the string

For MQFMT_ADMIN messages, if the specified string is shorter than the standard length of the *parameter,* the omitted characters are assumed to be nulls. If the specified string is longer than the standard length, it is an error.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For other programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFBS in a larger structure, and declare additional fields following MQCFBS, to represent the *String* field as required.

C language declaration

```
typedef struct tagMQCFBS {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  StringLength; /* Length of string */
  MQBYTE  String[1];    /* String value - first byte */

   } MQCFBS;
```

COBOL language declaration

```
**   MQCFBS structure
  10 MQCFBS.
**    Structure type
   15 MQCFBS-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFBS-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFBS-PARAMETER   PIC S9(9) BINARY.
**    Length of string
   15 MQCFBS-STRINGLENGTH PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFBS based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 StringLength fixed bin(31)  /* Length of string */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFBS                        DSECT
MQCFBS_TYPE                   DS   F        Structure type
MQCFBS_STRUCLENGTH            DS   F        Structure length
MQCFBS_PARAMETER              DS   F        Parameter identifier
MQCFBS_STRINGLENGTH           DS   F        Length of string
                              ORG  MQCFBS
MQCFBS_AREA                   DS   CL(MQCFBS_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFBS
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  StringLength As Long ' Operator identifier
  String as 1          ' String value - first byte
  End Type

Global MQCFBS_DEFAULT As MQCFBS
```

RPG language declaration (IBM i only)

```
D* MQCFBS Structure
D*
D* Structure type
D  BSTYP                  1      4I 0 INZ(3)
D* Structure length
D  BSLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  BSPRM                  9     12I 0 INZ(0)
D* Length of string
D  BSSTL                 13     16I 0 INZ(0)
D* String value - first byte
D  BSSRA                 17     16
D*
```

MQCFIF - PCF integer filter parameter:

The MQCFIF structure describes an integer filter parameter. The format name in the message descriptor is MQFMT_ADMIN.

The MQCFIF structure is used in Inquire commands to provide a filter condition. This filter condition is used to filter the results of the Inquire command and return to the user only those objects that satisfy the filter condition.

When an MQCFIF structure is present, the Version field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_3 or higher.

Fields for MQCFIF

Type **(MQLONG)**
Structure type.

This indicates that the structure is an MQCFIF structure describing an integer filter parameter. The value must be:

MQCFT_INTEGER_FILTER
Structure defining an integer filter.

StrucLength **(MQLONG)**
Structure length.

This is the length in bytes of the MQCFIF structure. The value must be:

MQCFIF_STRUC_LENGTH
Length of command format integer-parameter structure.

Parameter **(MQLONG)**
Parameter identifier.

This identifies the parameter that is to be filtered on. The value of this identifier depends on the parameter to be filtered on. Any of the parameters which can be used in the Inquire command can be used in this field.

The parameter is from the following groups of parameters:

- MQIA_*
- MQIACF_*
- MQIAMO_*
- MQIACH_*

Operator **(MQLONG)**
Operator identifier.

This identifies the operator that is being used to evaluate whether the parameter satisfies the filter-value.

Possible values are:

MQCFOP_GREATER
Greater than

MQCFOP_LESS
Less than

MQCFOP_EQUAL
Equal to

MQCFOP_NOT_EQUAL
Not equal to

MQCFOP_NOT_LESS
Greater than or equal to

MQCFOP_NOT_GREATER
Less than or equal to

MQCFOP_CONTAINS
Contains a specified value. Use MQCFOP_CONTAINS when filtering on lists of values or integers.

MQCFOP_EXCLUDES
Does not contain a specified value. Use MQCFOP_EXCLUDES when filtering on lists of values or integers.

See the *FilterValue* description for details telling you which operators can be used in which circumstances.

FilterValue **(MQLONG)**
Filter value identifier.

This specifies the filter-value that must be satisfied.

Depending on the parameter, the value and the permitted operators can be:

- An explicit integer value, if the parameter takes a single integer value.

  You can only use the following operators:
  - MQCFOP_GREATER
  - MQCFOP_LESS
  - MQCFOP_EQUAL
  - MQCFOP_NOT_EQUAL
  - MQCFOP_NOT_GREATER
  - MQCFOP_NOT_LESS
- An MQ constant, if the parameter takes a single value from a possible set of values (for example, the value MQCHT_SENDER on the *ChannelType* parameter). You can only use MQCFOP_EQUAL or MQCFOP_NOT_EQUAL.
- An explicit value or an MQ constant, as the case might be, if the parameter takes a list of values. You can use either MQCFOP_CONTAINS or MQCFOP_EXCLUDES. For example, if the value 6 is specified with the operator MQCFOP_CONTAINS, all items where one of the parameter values is 6 are listed.

For example, if you need to filter on queues that are enabled for put operations in your Inquire Queue command, the parameter would be MQIA_INHIBIT_PUT and the filter-value would be MQQA_PUT_ALLOWED.

The filter value must be a valid value for the parameter being tested.

C language declaration

```
typedef struct tagMQCFIF {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Operator;     /* Operator identifier */
  MQLONG  FilterValue;  /* Filter value */
 } MQCFIF;
```

COBOL language declaration

```
**   MQCFIF structure
  10 MQCFIF.
**    Structure type
   15 MQCFIF-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIF-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIF-PARAMETER   PIC S9(9) BINARY.
**    Operator identifier
   15 MQCFIF-OPERATOR    PIC S9(9) BINARY.
**    Filter value
   15 MQCFIF-FILTERVALUE PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFIF based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Operator    fixed bin(31)  /* Operator identifier */
  3 FilterValue fixed bin(31); /* Filter value */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFIF                          DSECT
MQCFIF_TYPE                     DS   F        Structure type
MQCFIF_STRUCLENGTH              DS   F        Structure length
MQCFIF_PARAMETER                DS   F        Parameter identifier
MQCFIF_OPERATOR                 DS   F        Operator identifier
MQCFIF_FILTERVALUE              DS   F        Filter value
MQCFIF_LENGTH                   EQU  *-MQCFIF Length of structure
                                ORG  MQCFIF
MQCFIF_AREA                     DS   CL(MQCFIF_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFIF
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Operator As Long     ' Operator identifier
  FilterValue As Long  ' Filter value
End Type

Global MQCFIF_DEFAULT As MQCFIF
```

RPG language declaration (IBM i only)

```
D* MQCFIF Structure
D*
D* Structure type
D  FIFTYP                 1      4I 0 INZ(3)
D* Structure length
D  FIFLEN                 5      8I 0 INZ(16)
D* Parameter identifier
D  FIFPRM                 9     12I 0 INZ(0)
```

```
D* Operator identifier
D  FIFOP                  13     16I 0 INZ(0)
D* Condition identifier
D  FIFFV                  17     20I 0 INZ(0)
D*
```

MQCFIL - PCF integer list parameter:

The MQCFIL structure describes an integer-list parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFIL structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *Count*, and *Values* fields to the values appropriate to the data.

The structure ends with a variable-length array of integers; see the *Values* field in the following section for further details.

Fields for MQCFIL

Type **(MQLONG)**
Structure type.

This indicates that the structure is an MQCFIL structure describing an integer-list parameter. The value must be:

MQCFT_INTEGER_LIST
Structure defining an integer list.

StrucLength **(MQLONG)**
Structure length.

This is the length in bytes of the MQCFIL structure, including the array of integers at the end of the structure (the *Values* field). The length must be a multiple of four, and must be sufficient to contain the array; any bytes between the end of the array and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *Values* field:

MQCFIL_STRUC_LENGTH_FIXED
Length of fixed part of command format integer-list parameter structure.

Parameter **(MQLONG)**
Parameter identifier.

This identifies the parameter with values that are contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1215 for details.

The parameter is from the following groups of parameters:

- MQIA_*
- MQIACF_*
- MQIAMO_*
- MQIACH_*

Count **(MQLONG)**
Count of parameter values.

This is the number of elements in the *Values* array; it must be zero or greater.

Values **(MQLONG×*Count*)**
Parameter values.

This is an array of values for the parameter identified by the *Parameter* field. For example, for MQIACF_Q_ATTRS, this field is a list of attribute selectors (MQCA_* and MQIA_* values).

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIL in a larger structure, and declare additional fields following MQCFIL, to represent the *Values* field as required.

C language declaration

```
typedef struct tagMQCFIL {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQLONG  Values[1];    /* Parameter values - first element */
 } MQCFIL;
```

COBOL language declaration

```
**   MQCFIL structure
  10 MQCFIL.
**    Structure type
   15 MQCFIL-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFIL based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31); /* Count of parameter values */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFIL                        DSECT
MQCFIL_TYPE                   DS   F        Structure type
MQCFIL_STRUCLENGTH            DS   F        Structure length
MQCFIL_PARAMETER              DS   F        Parameter identifier
MQCFIL_COUNT                  DS   F        Count of parameter values
MQCFIL_LENGTH                 EQU  *-MQCFIL Length of structure
                              ORG  MQCFIL
MQCFIL_AREA                   DS   CL(MQCFIL_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFIL
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
```

```
  Count As Long        ' Count of parameter values
End Type

Global MQCFIL_DEFAULT As MQCFIL
```

RPG language declaration (IBM i only)

```
D* MQCFIL Structure
D*
D* Structure type
D  ILTYP                  1      4I 0 INZ(5)
D* Structure length
D  ILLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  ILPRM                  9     12I 0 INZ(0)
D* Count of parameter values
D  ILCNT                 13     16I 0 INZ(0)
D*
```

MQCFIN - PCF integer parameter:

The MQCFIN structure describes an integer parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFIN structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *Value* field to the value appropriate to the data.

Fields for MQCFIN

Type **(MQLONG)**
Structure type.

This indicates that the structure is an MQCFIN structure describing an integer parameter. The value must be:

MQCFT_INTEGER
Structure defining an integer.

StrucLength **(MQLONG)**
Structure length.

This is the length in bytes of the MQCFIN structure. The value must be:

MQCFIN_STRUC_LENGTH
Length of command format integer-parameter structure.

Parameter **(MQLONG)**
Parameter identifier.

This identifies the parameter with a value that is contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1215 for details.

The parameter is from the following groups of parameters:

- MQIA_*
- MQIACF_*
- MQIAMO_*
- MQIACH_*

Value **(MQLONG)**
Parameter value.

This is the value of the parameter identified by the *Parameter* field.

C language declaration

```
typedef struct tagMQCFIN {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Value;        /* Parameter value */
 } MQCFIN;
```

COBOL language declaration

```
**   MQCFIN structure
  10 MQCFIN.
**    Structure type
   15 MQCFIN-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIN-PARAMETER   PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN-VALUE       PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFIN based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Value       fixed bin(31); /* Parameter value */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFIN                        DSECT
MQCFIN_TYPE                   DS   F        Structure type
MQCFIN_STRUCLENGTH            DS   F        Structure length
MQCFIN_PARAMETER              DS   F        Parameter identifier
MQCFIN_VALUE                  DS   F        Parameter value
MQCFIN_LENGTH                 EQU  *-MQCFIN Length of structure
                              ORG  MQCFIN
MQCFIN_AREA                   DS   CL(MQCFIN_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFIN
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Value As Long        ' Parameter value
End Type

Global MQCFIN_DEFAULT As MQCFIN
```

RPG language declaration (IBM i only)

```
D* MQCFIN Structure
D*
D* Structure type
D  INTYP                  1      4I 0 INZ(3)
D* Structure length
D  INLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  INPRM                  9     12I 0 INZ(0)
D* Parameter value
D  INVAL                 13     16I 0 INZ(0)
D*
```

MQCFSF - PCF string filter parameter:

The MQCFSF structure describes a string filter parameter. The format name in the message descriptor is MQFMT_ADMIN.

The MQCFSF structure is used in Inquire commands to provide a filter condition. This filter condition is used to filter the results of the Inquire command and return to the user only those objects that satisfy the filter condition.

The results of filtering character strings on EBCDIC-based systems might be different from those results achieved on ASCII-based systems. This difference is because comparison of character strings is based on the collating sequence of the internal built-in values representing the characters.

When an MQCFSF structure is present, the Version field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_3 or higher.

Fields for MQCFSF

Type **(MQLONG)**
Structure type.

This indicates that the structure is an MQCFSF structure describing a string filter parameter. The value must be:

MQCFT_STRING_FILTER
Structure defining a string filter.

StrucLength **(MQLONG)**
Structure length.

This is the length in bytes of the MQCFSF structure. The value must be:

MQCFSF_STRUC_LENGTH
MQCFSF_STRUC_LENGTH is the length, in bytes, of the MQCFSF structure, including the string at the end of the structure (the *FilterValue* field). The length must be a multiple of 4, and must be sufficient to contain the string. Bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *FilterValue* field:

MQCFSF_STRUC_LENGTH_FIXED
Length of fixed part of command format filter string-parameter structure.

Parameter **(MQLONG)**
Parameter identifier.

This identifies the parameter that is to be filtered on. The value of this identifier depends on the parameter to be filtered on. Any of the parameters which can be used in the Inquire command can be used in this field.

The parameter is from the following groups of parameters:

- MQCA_*
- MQCACF_*
- MQCAMO_*
- MQCACH_*

Operator **(MQLONG)**
Operator identifier.

This identifies the operator that is being used to evaluate whether the parameter satisfies the filter-value.

Possible values are:

MQCFOP_GREATER
Greater than

MQCFOP_LESS
Less than

MQCFOP_EQUAL
Equal to

MQCFOP_NOT_EQUAL
Not equal to

MQCFOP_NOT_LESS
Greater than or equal to

MQCFOP_NOT_GREATER
Less than or equal to

MQCFOP_LIKE
Matches a generic string

MQCFOP_NOT_LIKE
Does not match a generic string

MQCFOP_CONTAINS
Contains a specified string. Use MQCFOP_CONTAINS when filtering on lists of strings.

MQCFOP_EXCLUDES
Does not contain a specified string. Use MQCFOP_EXCLUDES when filtering on lists of strings.

MQCFOP_CONTAINS_GEN
Contains an item which matches a generic string. Use MQCFOP_CONTAINS_GEN when filtering on lists of strings.

MQCFOP_EXCLUDES_GEN
Does not contain any item which matches a generic string. Use MQCFOP_EXCLUDES_GEN when filtering on lists of strings.

See the *FilterValue* description for details telling you which operators can be used in which circumstances.

CodedCharSetId **(MQLONG)**
Coded character set identifier.

This specifies the coded character set identifier of the data in the *FilterValue* field. The following special value can be used:

MQCCSI_DEFAULT
Default character set identifier.

The string data is in the character set defined by the *CodedCharSetId* field in the MQ header structure that *precedes* the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH structure is at the start of the message.

FilterValueLength **(MQLONG)**
Length of filter-value string.

This is the length, in bytes, of the data in the *FilterValue* field. This parameter must be zero or greater, and does not need to be a multiple of 4.

FilterValue **(MQCHAR×***FilterValueLength***)**
Filter value.

This specifies the filter-value that must be satisfied. Depending on the parameter, the value and the permitted operators can be:

- An explicit string value.

  You can only use the following operators:
  - MQCFOP_GREATER
  - MQCFOP_LESS
  - MQCFOP_EQUAL
  - MQCFOP_NOT_EQUAL
  - MQCFOP_NOT_GREATER
  - MQCFOP_NOT_LESS
- A generic string value. This field is a character string with an asterisk at the end, for example ABC*. The operator must be either MQCFOP_LIKE or MQCFOP_NOT_LIKE. The characters must be valid for the attribute you are testing. If the operator is MQCFOP_LIKE, all items where the attribute value begins with the string (ABC in the example) are listed. If the operator is MQCFOP_NOT_LIKE, all items where the attribute value does not begin with the string are listed.
- If the parameter takes a list of string values, the operator can be:
  - MQCFOP_CONTAINS
  - MQCFOP_EXCLUDES
  - MQCFOP_CONTAINS_GEN
  - MQCFOP_EXCLUDES_GEN

  An item in a list of values. The value can be explicit or generic. If it is explicit, use MQCFOP_CONTAINS or MQCFOP_EXCLUDES as the operator. For example, if the value DEF is specified with the operator MQCFOP_CONTAINS, all items where one of the attribute values is DEF are listed. If it is generic, use MQCFOP_CONTAINS_GEN or MQCFOP_EXCLUDES_GEN as the operator. If ABC* is specified with the operator MQCFOP_CONTAINS_GEN, all items where one of the attribute values begins with ABC are listed.

Note:

1. If the specified string is shorter than the standard length of the parameter in MQFMT_ADMIN command messages, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.
2. When the queue manager reads an MQCFSF structure in an MQFMT_ADMIN message from the command input queue, the queue manager processes the string as though it had been specified on an MQI call. This processing means that within the string, the first null and the characters following it (up to the end of the string) are treated as blanks.

The filter value must be a valid value for the parameter being tested.

C language declaration

```
typedef struct tagMQCFSF {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Operator;     /* Operator identifier */
  MQLONG  CodedCharSetId; /* Coded character set identifier */
  MQLONG  FilterValueLength /* Filtervalue length */
  MQCHAR[1]  FilterValue;  /* Filter value */
 } MQCFSF;
```

COBOL language declaration

```
**   MQCFSF structure
  10 MQCFSF.
**    Structure type
   15 MQCFSF-TYPE        PIC S9(9) BINARY.
```

```
**    Structure length
   15 MQCFSF-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFSF-PARAMETER   PIC S9(9) BINARY.
**    Operator identifier
   15 MQCFSF-OPERATOR PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFSF-CODEDCHARSETID PIC S9(9) BINARY.
**    Filter value length
   15 MQCFSF-FILTERVALUE PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFSF based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Operator    fixed bin(31)  /* Operator identifier */
  3 CodedCharSetId    fixed bin(31)  /* Coded character set identifier */
  3 FilterValueLength fixed bin(31); /* Filter value length */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFSF                        DSECT
MQCFSF_TYPE                   DS   F        Structure type
MQCFSF_STRUCLENGTH            DS   F        Structure length
MQCFSF_PARAMETER              DS   F        Parameter identifier
MQCFSF_OPERATOR               DS   F        Operator identifier
MQCFSF_CODEDCHARSETID         DS   F        Coded character set identifier
MQCFSF_FILTERVALUELENGTH      DS   F        Filter value length
MQCFSF_LENGTH                 EQU  *-MQCFSF Length of structure
                              ORG  MQCFSF
MQCFSF_AREA                   DS   CL(MQCFSF_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFSF
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Operator As Long     ' Operator identifier
  CodedCharSetId As Long     ' Coded character set identifier
  FilterValueLength As Long  ' Operator identifier
  FilterValue As String*1  ' Condition value -- first character
End Type

Global MQCFSF_DEFAULT As MQCFSF
```

RPG language declaration (IBM i only)

```
D* MQCFSF Structure
D*
D* Structure type
D  FISTYP                  1      4I 0 INZ(3)
D* Structure length
D  FSFLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  FSFPRM                  9     12I 0 INZ(0)
D* Reserved field
D  FSFRSV                 13     16I 0 INZ(0)
D* Parameter value
D  FSFVAL                 17     16
D* Structure type
D  FSFTYP                 17     20I 0
D* Structure length
D  FSFLEN                 21     24I 0
D* Parameter value
```

```
D  FSFPRM                25     28I 0
D* Operator identifier
D  FSFOP                 29     32I 0
D* Coded character set identifier
D  FSFCSI                33     36I 0
D* Length of condition
D  FSFFVL                37     40 0
D* Condition value -- first character
D  FSFFV                 41     41
D*
```

MQCFSL - PCF string list parameter:

The MQCFSL structure describes a string-list parameter in a message which is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFSL structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *Count*, *StringLength*, and *Strings* fields to the values appropriate to the data.

The structure ends with a variable-length array of character strings; see the *Strings* field section for further details.

See "Usage notes" on page 1215 for further information about how to use the structure.

Fields for MQCFSL

Type **(MQLONG)**
Structure type.

This indicates that the structure is an MQCFSL structure describing a string-list parameter. The value must be:

MQCFT_STRING_LIST
Structure defining a string list.

StrucLength **(MQLONG)**
Structure length.

This is the length in bytes of the MQCFSL structure, including the data at the end of the structure (the *Strings* field). The length must be a multiple of four, and must be sufficient to contain all the strings; any bytes between the end of the strings and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *Strings* field:

MQCFSL_STRUC_LENGTH_FIXED
Length of fixed part of command format string-list parameter structure.

Parameter **(MQLONG)**
Parameter identifier.

This identifies the parameter with values that are contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1215 for details.

The parameter is from the following groups of parameters:

- MQCA_*
- MQCACF_*

- MQCAMO_*
- MQCACH_*

CodedCharSetId **(MQLONG)**
Coded character set identifier.

This specifies the coded character set identifier of the data in the *Strings* field. The following special value can be used:

MQCCSI_DEFAULT
Default character set identifier.

The string data is in the character set defined by the *CodedCharSetId* field in the MQ header structure that *precedes* the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH structure is at the start of the message.

Count **(MQLONG)**
Count of parameter values.

This is the number of strings present in the *Strings* field; it must be zero or greater.

StringLength **(MQLONG)**
Length of one string.

This is the length in bytes of one parameter value, that is the length of one string in the *Strings* field; all the strings are this length. The length must be zero or greater, and need not be a multiple of four.

Strings **(MQCHAR**×*StringLength*×*Count***)**
String values.

This is a set of string values for the parameter identified by the *Parameter* field. The number of strings is given by the *Count* field, and the length of each string is given by the *StringLength* field. The strings are concatenated together, with no bytes skipped between adjacent strings. The total length of the strings is the length of one string multiplied by the number of strings present (that is, *StringLength*×*Count*).

- In MQFMT_ADMIN command messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.
- In MQFMT_ADMIN response messages, string parameters might be returned padded with blanks to the standard length of the parameter.
- In MQFMT_EVENT messages, trailing blanks might be omitted from string parameters (that is, the string might be shorter than the standard length of the parameter).

In all cases, *StringLength* gives the length of the string present in the message.

The strings can contain any characters that are in the character set defined by *CodedCharSetId*, and that are valid for the parameter identified by *Parameter*.

Note: When the queue manager reads an MQCFSL structure in an MQFMT_ADMIN message from the command input queue, the queue manager processes each string in the list as though it had been specified on an MQI call. This processing means that within each string, the first null, and the characters following it (up to the end of the string) are treated as blanks.

In responses and all other cases, a null character in a string is treated as normal data, and does not act as a delimiter for the string. This treatment means that when a receiving application reads a MQFMT_PCF, MQFMT_EVENT, or MQFMT_ADMIN message, the receiving application receives all the data specified by the sending application.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.

- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFSL in a larger structure, and declare additional fields following MQCFSL, to represent the *Strings* field as required.

C language declaration

```
typedef struct tagMQCFSL {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQLONG  Count;           /* Count of parameter values */
  MQLONG  StringLength;    /* Length of one string */
  MQCHAR  Strings[1];      /* String values - first
                              character */
 } MQCFSL;
```

COBOL language declaration

```
**   MQCFSL structure
  10 MQCFSL.
**    Structure type
   15 MQCFSL-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFSL-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFSL-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFSL-CODEDCHARSETID PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFSL-COUNT          PIC S9(9) BINARY.
**    Length of one string
   15 MQCFSL-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFSL based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 Count          fixed bin(31), /* Count of parameter values */
  3 StringLength   fixed bin(31); /* Length of one string */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFSL                      DSECT
MQCFSL_TYPE                 DS   F        Structure type
MQCFSL_STRUCLENGTH          DS   F        Structure length
MQCFSL_PARAMETER            DS   F        Parameter identifier
MQCFSL_CODEDCHARSETID       DS   F        Coded character set
*                                         identifier
MQCFSL_COUNT                DS   F        Count of parameter values
MQCFSL_STRINGLENGTH         DS   F        Length of one string
MQCFSL_LENGTH               EQU  *-MQCFSL Length of structure
                            ORG  MQCFSL
MQCFSL_AREA                 DS   CL(MQCFSL_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFSL
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  CodedCharSetId As Long  ' Coded character set identifier
```

```
  Count As Long           ' Count of parameter values
  StringLength As Long    ' Length of one string
End Type

Global MQCFSL_DEFAULT As MQCFSL
```

RPG language declaration (IBM i only)

```
D* MQCFSL Structure
D*
D* Structure type
D  SLTYP                  1      4I 0 INZ(6)
D* Structure length
D  SLLEN                  5      8I 0 INZ(24
D* Parameter identifier
D  SLPRM                  9     12I 0 INZ(0)
D* Coded character set identifier
D  SLCSI                 13     16I 0 INZ(0)
D* Count of parameter values
D  SLCNT                 17     20I 0 INZ(0)
D* Length of one string
D  SLSTL                 21     24I 0 INZ(0)
```

MQCFST - PCF string parameter:

The MQCFST structure describes a string parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFST structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see Message descriptor for a PCF command). Also in this case, not all the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *StringLength*, and *String* fields to the values appropriate to the data.

The structure ends with a variable-length character string; see the *String* field section for further details.

See "Usage notes" on page 1215 for further information about how to use the structure.

Fields for MQCFST

Type **(MQLONG)**
Structure type.

This indicates that the structure is an MQCFST structure describing a string parameter. The value must be:

MQCFT_STRING
Structure defining a string.

StrucLength **(MQLONG)**
Structure length.

This is the length in bytes of the MQCFST structure, including the string at the end of the structure (the *String* field). The length must be a multiple of four, and must be sufficient to contain the string; any bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *String* field:

MQCFST_STRUC_LENGTH_FIXED
Length of fixed part of command format string-parameter structure.

Parameter **(MQLONG)**
Parameter identifier.

This identifies the parameter with a value that is contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see "MQCFH - PCF header" on page 1215 for details.

The parameter is from the following groups of parameters:

- MQCA_*
- MQCACF_*
- MQCAMO_*
- MQCACH_*

CodedCharSetId **(MQLONG)**
Coded character set identifier.

This specifies the coded character set identifier of the data in the *String* field. The following special value can be used:

MQCCSI_DEFAULT
Default character set identifier.

The string data is in the character set defined by the *CodedCharSetId* field in the MQ header structure that *precedes* the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH structure is at the start of the message.

StringLength **(MQLONG)**
Length of string.

This is the length in bytes of the data in the *String* field; it must be zero or greater. This length does not need to be a multiple of four.

String **(MQCHAR**×*StringLength***)**
String value.

This is the value of the parameter identified by the *Parameter* field:

- In MQFMT_ADMIN command messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, it is an error.
- In MQFMT_ADMIN response messages, string parameters might be returned padded with blanks to the standard length of the parameter.
- In MQFMT_EVENT messages, trailing blanks might be omitted from string parameters (that is, the string can be shorter than the standard length of the parameter).

The value of *StringLength* depends on whether, when the specified string is shorter than the standard length, padding blanks have been added to the string. If so, the value of *StringLength* is the sum of the actual length of the string plus the padded blanks.

The string can contain any characters that are in the character set defined by *CodedCharSetId*, and that are valid for the parameter identified by *Parameter*.

Note: When the queue manager reads an MQCFST structure in an MQFMT_ADMIN message from the command input queue, the queue manager processes the string as though it had been specified on an MQI call. This processing means that within the string, the first null and the characters following it (up to the end of the string) are treated as blanks.

In responses and all other cases, a null character in the string is treated as normal data, and does not act as a delimiter for the string. This treatment means that when a receiving application reads a MQFMT_PCF, MQFMT_EVENT, or MQFMT_ADMIN message, the receiving application receives all the data specified by the sending application.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user must include MQCFST in a larger structure, and declare an additional field or additional fields following MQCFST, to represent the *String* field as required.

C language declaration

```
typedef struct tagMQCFST {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQLONG  StringLength;    /* Length of string */
  MQCHAR  String[1];       /* String value - first
                              character */
 } MQCFST;
```

COBOL language declaration

```
**   MQCFST structure
  10 MQCFST.
**    Structure type
   15 MQCFST-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFST-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFST-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFST-CODEDCHARSETID PIC S9(9) BINARY.
**    Length of string
   15 MQCFST-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFST based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 StringLength   fixed bin(31); /* Length of string */
```

System/390 assembler-language declaration (z/OS only)

```
MQCFST                      DSECT
MQCFST_TYPE                 DS   F        Structure type
MQCFST_STRUCLENGTH          DS   F        Structure length
MQCFST_PARAMETER            DS   F        Parameter identifier
MQCFST_CODEDCHARSETID       DS   F        Coded character set
*                                         identifier
MQCFST_STRINGLENGTH         DS   F        Length of string
MQCFST_LENGTH               EQU  *-MQCFST Length of structure
                            ORG  MQCFST
MQCFST_AREA                 DS   CL(MQCFST_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFST
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  CodedCharSetId As Long  ' Coded character set identifier
```

```
  StringLength As Long    ' Length of string
End Type

Global MQCFST_DEFAULT As MQCFST
```

RPG language declaration (IBM i only)

```
D* MQCFST Structure
D*
D* Structure type
D  STTYP                  1      4I 0 INZ(4)
D* Structure length
D  STLEN                  5      8I 0 INZ(20)
D* Parameter identifier
D  STPRM                  9     12I 0 INZ(0)
D* Coded character set identifier
D  STCSI                 13     16I 0 INZ(0)
D* Length of string
D  STSTL                 17     20I 0 INZ(0)
D*
```

PCF example

The compiled program, written in C language, in the example uses WebSphere MQ for Windows. It inquires of the default queue manager about a subset of the attributes for all local queues defined to it. It then produces an output file, SAVEQMGR.TST, in the directory from which it was run for use with RUNMQSC.

Inquire local queue attributes

This following section provides an example of how Programmable Command Formats can be used in a program for administration of WebSphere MQ queues.

The program is given as an example of using PCFs and has been limited to a simple case. This program is of most use as an example if you are considering the use of PCFs to manage your WebSphere MQ environment.

Program listing

```
/*============================================================================*/
/*                                                                            */
/* This is a program to inquire of the default queue manager about the        */
/* local queues defined to it.                                                */
/*                                                                            */
/* The program takes this information and appends it to a file                */
/* SAVEQMGR.TST which is of a format suitable for RUNMQSC. It could,          */
/* therefore, be used to recreate or clone a queue manager.                   */
/*                                                                            */
/* It is offered as an example of using Programmable Command Formats (PCFs)   */
/* as a method for administering a queue manager.                             */
/*                                                                            */
/*============================================================================*/

/* Include standard libraries */
#include <memory.h>
#include <stdio.h>

/* Include MQSeries headers */
#include <cmqc.h>
#include <cmqcfc.h>
#include <cmqxc.h>

typedef struct LocalQParms {
   MQCHAR48    QName;
   MQLONG      QType;
   MQCHAR64    QDesc;
```

```
   MQLONG      InhibitPut;
   MQLONG      DefPriority;
   MQLONG      DefPersistence;
   MQLONG      InhibitGet;
   MQCHAR48    ProcessName;
   MQLONG      MaxQDepth;
   MQLONG      MaxMsgLength;
   MQLONG      BackoutThreshold;
   MQCHAR48    BackoutReqQName;
   MQLONG      Shareability;
   MQLONG      DefInputOpenOption;
   MQLONG      HardenGetBackout;
   MQLONG      MsgDeliverySequence;
   MQLONG      RetentionInterval;
   MQLONG      DefinitionType;
   MQLONG      Usage;
   MQLONG      OpenInputCount;
   MQLONG      OpenOutputCount;
   MQLONG      CurrentQDepth;
   MQCHAR12    CreationDate;
   MQCHAR8     CreationTime;
   MQCHAR48    InitiationQName;
   MQLONG      TriggerControl;
   MQLONG      TriggerType;
   MQLONG      TriggerMsgPriority;
   MQLONG      TriggerDepth;
   MQCHAR64    TriggerData;
   MQLONG      Scope;
   MQLONG      QDepthHighLimit;
   MQLONG      QDepthLowLimit;
   MQLONG      QDepthMaxEvent;
   MQLONG      QDepthHighEvent;
   MQLONG      QDepthLowEvent;
   MQLONG      QServiceInterval;
   MQLONG      QServiceIntervalEvent;
} LocalQParms;

MQOD  ObjDesc = { MQOD_DEFAULT };
MQMD  md      = { MQMD_DEFAULT  };
MQPMO pmo     = { MQPMO_DEFAULT };
MQGMO gmo     = { MQGMO_DEFAULT };

void ProcessStringParm( MQCFST *pPCFString, LocalQParms *DefnLQ );

void ProcessIntegerParm( MQCFIN *pPCFInteger, LocalQParms *DefnLQ );

void AddToFileQLOCAL( LocalQParms DefnLQ );

void MQParmCpy( char *target, char *source, int length );

void PutMsg( MQHCONN   hConn       /* Connection to queue manager            */
           , MQCHAR8   MsgFormat   /* Format of user data to be put in msg   */
           , MQHOBJ    hQName      /* handle of queue to put the message to  */
           , MQCHAR48  QName       /* name of queue to put the message to    */
           , MQBYTE   *UserMsg     /* The user data to be put in the message */
           , MQLONG    UserMsgLen  /*                                        */
           );

void GetMsg( MQHCONN   hConn          /* handle of queue manager            */
           , MQLONG    MQParm         /* Options to specify nature of get   */
           , MQHOBJ    hQName         /* handle of queue to read from       */
           , MQBYTE   *UserMsg        /* Input/Output buffer containing msg */
           , MQLONG    ReadBufferLen  /* Length of supplied buffer          */
           );
MQHOBJ OpenQ( MQHCONN     hConn
            , MQCHAR48    QName
            , MQLONG      OpenOpts
```

```
          );

int main( int argc, char *argv[] )
{
  MQCHAR48             QMgrName;         /* Name of connected queue mgr     */
  MQHCONN              hConn;            /* handle to connected queue mgr   */
  MQOD                 ObjDesc;          /*                                 */
  MQLONG               OpenOpts;         /*                                 */
  MQLONG               CompCode;         /* MQ API completion code          */
  MQLONG               Reason;           /* Reason qualifying above         */
                                         /*                                 */
  MQHOBJ               hAdminQ;          /* handle to output queue          */
  MQHOBJ               hReplyQ;          /* handle to input queue           */
                                         /*                                 */
  MQLONG               AdminMsgLen;      /* Length of user message buffer   */
  MQBYTE              *pAdminMsg;        /* Ptr to outbound data buffer     */
  MQCFH               *pPCFHeader;       /* Ptr to PCF header structure     */
  MQCFST              *pPCFString;       /* Ptr to PCF string parm block    */
  MQCFIN              *pPCFInteger;      /* Ptr to PCF integer parm block   */
  MQLONG              *pPCFType;         /* Type field of PCF message parm  */
  LocalQParms          DefnLQ;           /*                                 */
                                         /*                                 */
  char                 ErrorReport[40];  /*                       */
  MQCHAR8              MsgFormat;        /* Format of inbound message       */
  short                Index;            /* Loop counter                    */

  /* Connect to default queue manager */
  QMgrName[0] =  '\0';                   /* set to null   default QM */
  if ( argc > 1 )
    strcpy(QMgrName, argv[1]);

  MQCONN(  QMgrName                      /*  use default queue manager  */
        , &hConn                         /*  queue manager handle       */
        , &CompCode                      /*  Completion code            */
        , &Reason                        /*  Reason qualifying CompCode */
        );

  if ( CompCode != MQCC_OK ) {
     printf( "MQCONN failed for %s, CC=%d RC=%d\n"
           , QMgrName
           , CompCode
           , Reason
           );
     exit( -1 );
  } /* endif */

  /* Open all the required queues */
  hAdminQ = OpenQ( hConn, "SYSTEM.ADMIN.COMMAND.QUEUE\0", MQOO_OUTPUT );

  hReplyQ = OpenQ( hConn, "SAVEQMGR.REPLY.QUEUE\0", MQOO_INPUT_EXCLUSIVE );

  /* ************************************************************** */
  /* Put a message to the SYSTEM.ADMIN.COMMAND.QUEUE to inquire all     */
  /* the local queues defined on the queue manager.                     */
  /*                                                                    */
  /* The request consists of a Request Header and a parameter block     */
  /* used to specify the generic search. The header and the parameter   */
  /* block follow each other in a contiguous buffer which is pointed    */
  /* to by the variable pAdminMsg. This entire buffer is then put to    */
  /* the queue.                                                         */
  /*                                                                    */
  /* The command server, (use STRMQCSV to start it), processes the      */
  /* SYSTEM.ADMIN.COMMAND.QUEUE and puts a reply on the application     */
  /* ReplyToQ for each defined queue.                                   */
  /* ************************************************************** */

  /* Set the length for the message buffer */
```

```
AdminMsgLen = MQCFH_STRUC_LENGTH
            + MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH
            + MQCFIN_STRUC_LENGTH
            ;

/* ---------------------------------------------------------------- */
/* Set pointers to message data buffers                             */
/*                                                                  */
/* pAdminMsg points to the start of the message buffer              */
/*                                                                  */
/* pPCFHeader also points to the start of the message buffer. It is */
/* used to indicate the type of command we wish to execute and the  */
/* number of parameter blocks following in the message buffer.      */
/*                                                                  */
/* pPCFString points into the message buffer immediately after the  */
/* header and is used to map the following bytes onto a PCF string  */
/* parameter block. In this case the string is used to indicate the */
/* nameof the queue we want details about, * indicating all queues. */
/*                                                                  */
/* pPCFInteger points into the message buffer immediately after the */
/* string block described above. It is used to map the following    */
/* bytes onto a PCF integer parameter block. This block indicates   */
/* the type of queue we wish to receive details about, thereby      */
/* qualifying the generic search set up by passing the previous     */
/* string parameter.                                                */
/*                                                                  */
/* Note that this example is a generic search for all attributes of */
/* all local queues known to the queue manager. By using different, */
/* or more, parameter blocks in the request header it is possible   */
/* to narrow the search.                                            */
/* ---------------------------------------------------------------- */

pAdminMsg   = (MQBYTE *)malloc( AdminMsgLen );

pPCFHeader  = (MQCFH *)pAdminMsg;

pPCFString  = (MQCFST *)(pAdminMsg
                         + MQCFH_STRUC_LENGTH
                         );

pPCFInteger = (MQCFIN *)( pAdminMsg
                         + MQCFH_STRUC_LENGTH
                         + MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH
                         );


/* Setup request header */
pPCFHeader->Type           = MQCFT_COMMAND;
pPCFHeader->StrucLength    = MQCFH_STRUC_LENGTH;
pPCFHeader->Version        = MQCFH_VERSION_1;
pPCFHeader->Command        = MQCMD_INQUIRE_Q;
pPCFHeader->MsgSeqNumber   = MQCFC_LAST;
pPCFHeader->Control        = MQCFC_LAST;
pPCFHeader->ParameterCount = 2;

/* Setup parameter block */
pPCFString->Type           = MQCFT_STRING;
pPCFString->StrucLength    = MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH;
pPCFString->Parameter      = MQCA_Q_NAME;
pPCFString->CodedCharSetId = MQCCSI_DEFAULT;
pPCFString->StringLength   = MQ_Q_NAME_LENGTH;
memset( pPCFString->String, ' ', MQ_Q_NAME_LENGTH );
memcpy( pPCFString->String, "*", 1 );

/* Setup parameter block */
pPCFInteger->Type        = MQCFT_INTEGER;
pPCFInteger->StrucLength = MQCFIN_STRUC_LENGTH;
```

```
pPCFInteger->Parameter   = MQIA_Q_TYPE;
pPCFInteger->Value       = MQQT_LOCAL;

PutMsg(  hConn                    /* Queue manager handle            */
      ,  MQFMT_ADMIN              /* Format of message               */
      ,  hAdminQ                  /* Handle of command queue         */
      ,  "SAVEQMGR.REPLY.QUEUE\0" /* reply to queue                  */
      ,  (MQBYTE *)pAdminMsg      /* Data part of message to put     */
      ,  AdminMsgLen
      );

free( pAdminMsg );

/* ****************************************************************** */
/* Get and process the replies received from the command server onto  */
/* the applications ReplyToQ.                                         */
/*                                                                    */
/* There will be one message per defined local queue.                 */
/*                                                                    */
/* The last message will have the Control field of the PCF header     */
/* set to MQCFC_LAST. All others will be MQCFC_NOT_LAST.              */
/*                                                                    */
/* An individual Reply message consists of a header followed by a     */
/* number a parameters, the exact number, type and order will depend  */
/* upon the type of request.                                          */
/*                                                                    */
/* ------------------------------------------------------------------ */
/*                                                                    */
/* The message is retrieved into a buffer pointed to by pAdminMsg.    */
/* This buffer has been allocated enough memory to hold every         */
/* parameter needed for a local queue definition.                     */
/*                                                                    */
/* pPCFHeader is then allocated to point also to the beginning of     */
/* the buffer and is used to access the PCF header structure. The     */
/* header contains several fields. The one we are specifically        */
/* interested in is the ParameterCount. This tells us how many        */
/* parameters follow the header in the message buffer. There is       */
/* one parameter for each local queue attribute known by the          */
/* queue manager.                                                     */
/*                                                                    */
/* At this point we do not know the order or type of each parameter   */
/* block in the buffer, the first MQLONG of each block defines its    */
/* type; they may be parameter blocks containing either strings or    */
/* integers.                                                          */
/*                                                                    */
/* pPCFType is used initially to point to the first byte beyond the   */
/* known parameter block. Initially then, it points to the first byte */
/* after the PCF header. Subsequently it is incremented by the length */
/* of the identified parameter block and therefore points at the      */
/* next. Looking at the value of the data pointed to by pPCFType we   */
/* can decide how to process the next group of bytes, either as a     */
/* string, or an integer.                                             */
/*                                                                    */
/* In this way we parse the message buffer extracting the values of   */
/* each of the parameters we are interested in.                       */
/*                                                                    */
/* ****************************************************************** */

/* AdminMsgLen is to be set to the length of the expected reply       */
/* message. This structure is specific to Local Queues.               */
AdminMsgLen =   MQCFH_STRUC_LENGTH
            + ( MQCFST_STRUC_LENGTH_FIXED * 7 )
            + ( MQCFIN_STRUC_LENGTH       * 39 )
            + ( MQ_Q_NAME_LENGTH          * 6  )
            + ( MQ_Q_MGR_NAME_LENGTH      * 2  )
            +   MQ_Q_DESC_LENGTH
            +   MQ_PROCESS_NAME_LENGTH
```

```
            +   MQ_CREATION_DATE_LENGTH
            +   MQ_CREATION_TIME_LENGTH
            +   MQ_TRIGGER_DATA_LENGTH + 100
            ;

/* Set pointers to message data buffers */
pAdminMsg = (MQBYTE *)malloc( AdminMsgLen );

do {

   GetMsg(  hConn                     /* Queue manager handle           */
         ,  MQGMO_WAIT
         ,  hReplyQ                   /* Get queue handle               */
         ,  (MQBYTE *)pAdminMsg       /* pointer to message area        */
         ,  AdminMsgLen               /* length of get buffer           */
         );

   /* Examine Header */
   pPCFHeader = (MQCFH *)pAdminMsg;

   /* Examine first parameter */
   pPCFType = (MQLONG *)(pAdminMsg + MQCFH_STRUC_LENGTH);

   Index = 1;

   while ( Index <= pPCFHeader->ParameterCount ) {

      /* Establish the type of each parameter and allocate  */
      /* a pointer of the correct type to reference it.     */
      switch ( *pPCFType ) {
      case MQCFT_INTEGER:
         pPCFInteger = (MQCFIN *)pPCFType;
         ProcessIntegerParm( pPCFInteger, &DefnLQ );
         Index++;
         /* Increment the pointer to the next parameter by the */
         /* length of the current parm.                        */
         pPCFType = (MQLONG *)( (MQBYTE *)pPCFType
                              + pPCFInteger->StrucLength
                              );
         break;
      case MQCFT_STRING:
         pPCFString = (MQCFST *)pPCFType;
         ProcessStringParm( pPCFString, &DefnLQ );
         Index++;
         /* Increment the pointer to the next parameter by the */
         /* length of the current parm.                        */
         pPCFType = (MQLONG *)( (MQBYTE *)pPCFType
                              + pPCFString->StrucLength
                              );
         break;
      } /* endswitch */

   } /* endwhile */

   /* ********************************************************* */
   /* Message parsed, append to output file                     */
   /* ********************************************************* */
   AddToFileQLOCAL( DefnLQ );


   /* ********************************************************* */
   /* Finished processing the current message, do the next one. */
   /* ********************************************************* */

} while ( pPCFHeader->Control == MQCFC_NOT_LAST ); /* enddo */

free( pAdminMsg );
```

```
  /* ************************************** */
  /* Processing of the local queues complete */
  /* ************************************** */

}

void ProcessStringParm( MQCFST *pPCFString, LocalQParms *DefnLQ )
{
   switch ( pPCFString->Parameter ) {
   case MQCA_Q_NAME:
      MQParmCpy( DefnLQ->QName, pPCFString->String, 48 );
      break;
   case MQCA_Q_DESC:
      MQParmCpy( DefnLQ->QDesc, pPCFString->String, 64 );
      break;
   case MQCA_PROCESS_NAME:
      MQParmCpy( DefnLQ->ProcessName, pPCFString->String, 48 );
      break;
   case MQCA_BACKOUT_REQ_Q_NAME:
      MQParmCpy( DefnLQ->BackoutReqQName, pPCFString->String, 48 );
      break;
   case MQCA_CREATION_DATE:
      MQParmCpy( DefnLQ->CreationDate, pPCFString->String, 12 );
      break;
   case MQCA_CREATION_TIME:
      MQParmCpy( DefnLQ->CreationTime, pPCFString->String, 8 );
      break;
   case MQCA_INITIATION_Q_NAME:
      MQParmCpy( DefnLQ->InitiationQName, pPCFString->String, 48 );
      break;
   case MQCA_TRIGGER_DATA:
      MQParmCpy( DefnLQ->TriggerData, pPCFString->String, 64 );
      break;
   } /* endswitch */
}

void ProcessIntegerParm( MQCFIN *pPCFInteger, LocalQParms *DefnLQ )
{
   switch ( pPCFInteger->Parameter ) {
   case MQIA_Q_TYPE:
      DefnLQ->QType = pPCFInteger->Value;
      break;
   case MQIA_INHIBIT_PUT:
      DefnLQ->InhibitPut = pPCFInteger->Value;
      break;
   case MQIA_DEF_PRIORITY:
      DefnLQ->DefPriority = pPCFInteger->Value;
      break;
   case MQIA_DEF_PERSISTENCE:
      DefnLQ->DefPersistence = pPCFInteger->Value;
      break;
   case MQIA_INHIBIT_GET:
      DefnLQ->InhibitGet = pPCFInteger->Value;
      break;
   case MQIA_SCOPE:
      DefnLQ->Scope = pPCFInteger->Value;
      break;
   case MQIA_MAX_Q_DEPTH:
      DefnLQ->MaxQDepth = pPCFInteger->Value;
      break;
   case MQIA_MAX_MSG_LENGTH:
      DefnLQ->MaxMsgLength = pPCFInteger->Value;
      break;
   case MQIA_BACKOUT_THRESHOLD:
      DefnLQ->BackoutThreshold = pPCFInteger->Value;
      break;
```

```
   case MQIA_SHAREABILITY:
      DefnLQ->Shareability = pPCFInteger->Value;
      break;
   case MQIA_DEF_INPUT_OPEN_OPTION:
      DefnLQ->DefInputOpenOption = pPCFInteger->Value;
      break;
   case MQIA_HARDEN_GET_BACKOUT:
      DefnLQ->HardenGetBackout = pPCFInteger->Value;
      break;
   case MQIA_MSG_DELIVERY_SEQUENCE:
      DefnLQ->MsgDeliverySequence = pPCFInteger->Value;
      break;
   case MQIA_RETENTION_INTERVAL:
      DefnLQ->RetentionInterval = pPCFInteger->Value;
      break;
   case MQIA_DEFINITION_TYPE:
      DefnLQ->DefinitionType = pPCFInteger->Value;
      break;
   case MQIA_USAGE:
      DefnLQ->Usage = pPCFInteger->Value;
      break;
   case MQIA_OPEN_INPUT_COUNT:
      DefnLQ->OpenInputCount = pPCFInteger->Value;
      break;
   case MQIA_OPEN_OUTPUT_COUNT:
      DefnLQ->OpenOutputCount = pPCFInteger->Value;
      break;
   case MQIA_CURRENT_Q_DEPTH:
      DefnLQ->CurrentQDepth = pPCFInteger->Value;
      break;
   case MQIA_TRIGGER_CONTROL:
      DefnLQ->TriggerControl = pPCFInteger->Value;
      break;
   case MQIA_TRIGGER_TYPE:
      DefnLQ->TriggerType = pPCFInteger->Value;
      break;
   case MQIA_TRIGGER_MSG_PRIORITY:
      DefnLQ->TriggerMsgPriority = pPCFInteger->Value;
      break;
   case MQIA_TRIGGER_DEPTH:
      DefnLQ->TriggerDepth = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_HIGH_LIMIT:
      DefnLQ->QDepthHighLimit = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_LOW_LIMIT:
      DefnLQ->QDepthLowLimit = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_MAX_EVENT:
      DefnLQ->QDepthMaxEvent = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_HIGH_EVENT:
      DefnLQ->QDepthHighEvent = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_LOW_EVENT:
      DefnLQ->QDepthLowEvent = pPCFInteger->Value;
      break;
   case MQIA_Q_SERVICE_INTERVAL:
      DefnLQ->QServiceInterval = pPCFInteger->Value;
      break;
   case MQIA_Q_SERVICE_INTERVAL_EVENT:
      DefnLQ->QServiceIntervalEvent = pPCFInteger->Value;
      break;
   } /* endswitch */
}

/* ------------------------------------------------------------------------ */
```

```
/*                                                                  */
/* This process takes the attributes of a single local queue and adds them  */
/* to the end of a file, SAVEQMGR.TST, which can be found in the current    */
/* directory.                                                       */
/*                                                                  */
/* The file is of a format suitable for subsequent input to RUNMQSC.        */
/*                                                                  */
/* ---------------------------------------------------------------------- */
void AddToFileQLOCAL( LocalQParms DefnLQ )
{
   char    ParmBuffer[120];   /* Temporary buffer to hold for output to file */
   FILE   *fp;                /* Pointer to a file                          */

   /* Append these details to the end of the current SAVEQMGR.TST file */
   fp = fopen( "SAVEQMGR.TST", "a" );

   sprintf( ParmBuffer, "DEFINE QLOCAL ('%s') REPLACE +\n", DefnLQ.QName );
   fputs( ParmBuffer, fp );

   sprintf( ParmBuffer, "       DESCR('%s') +\n" , DefnLQ.QDesc );
   fputs( ParmBuffer, fp );

   if ( DefnLQ.InhibitPut == MQQA_PUT_ALLOWED ) {
      sprintf( ParmBuffer, "       PUT(ENABLED) +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       PUT(DISABLED) +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   sprintf( ParmBuffer, "       DEFPRTY(%d) +\n", DefnLQ.DefPriority );
   fputs( ParmBuffer, fp );

   if ( DefnLQ.DefPersistence == MQPER_PERSISTENT ) {
      sprintf( ParmBuffer, "       DEFPSIST(YES) +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       DEFPSIST(NO) +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   if ( DefnLQ.InhibitGet == MQQA_GET_ALLOWED ) {
      sprintf( ParmBuffer, "       GET(ENABLED) +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       GET(DISABLED) +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   sprintf( ParmBuffer, "       MAXDEPTH(%d) +\n", DefnLQ.MaxQDepth );
   fputs( ParmBuffer, fp );

   sprintf( ParmBuffer, "       MAXMSGL(%d) +\n", DefnLQ.MaxMsgLength );
   fputs( ParmBuffer, fp );

   if ( DefnLQ.Shareability == MQQA_SHAREABLE ) {
      sprintf( ParmBuffer, "       SHARE +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       NOSHARE +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   if ( DefnLQ.DefInputOpenOption == MQOO_INPUT_SHARED ) {
      sprintf( ParmBuffer, "       DEFSOPT(SHARED) +\n" );
      fputs( ParmBuffer, fp );
   } else {
```

```
   sprintf( ParmBuffer, "       DEFSOPT(EXCL) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.MsgDeliverySequence == MQMDS_PRIORITY ) {
   sprintf( ParmBuffer, "       MSGDLVSQ(PRIORITY) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       MSGDLVSQ(FIFO) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.HardenGetBackout == MQQA_BACKOUT_HARDENED ) {
   sprintf( ParmBuffer, "       HARDENBO +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       NOHARDENBO +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.Usage == MQUS_NORMAL ) {
   sprintf( ParmBuffer, "       USAGE(NORMAL) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       USAGE(XMIT) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.TriggerControl == MQTC_OFF ) {
   sprintf( ParmBuffer, "       NOTRIGGER +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       TRIGGER +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

switch ( DefnLQ.TriggerType ) {
case MQTT_NONE:
   sprintf( ParmBuffer, "       TRIGTYPE(NONE) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_FIRST:
   sprintf( ParmBuffer, "       TRIGTYPE(FIRST) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_EVERY:
   sprintf( ParmBuffer, "       TRIGTYPE(EVERY) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_DEPTH:
   sprintf( ParmBuffer, "       TRIGTYPE(DEPTH) +\n" );
   fputs( ParmBuffer, fp );
   break;
} /* endswitch */

sprintf( ParmBuffer, "       TRIGDPTH(%d) +\n", DefnLQ.TriggerDepth );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       TRIGMPRI(%d) +\n", DefnLQ.TriggerMsgPriority);
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       TRIGDATA('%s') +\n", DefnLQ.TriggerData );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       PROCESS('%s') +\n", DefnLQ.ProcessName );
fputs( ParmBuffer, fp );
```

```
sprintf( ParmBuffer, "       INITQ('%s') +\n", DefnLQ.InitiationQName );
fputs( ParmBuffer, fp );


sprintf( ParmBuffer, "       RETINTVL(%d) +\n", DefnLQ.RetentionInterval );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       BOTHRESH(%d) +\n", DefnLQ.BackoutThreshold );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       BOQNAME('%s') +\n", DefnLQ.BackoutReqQName );
fputs( ParmBuffer, fp );

if ( DefnLQ.Scope == MQSCO_Q_MGR ) {
   sprintf( ParmBuffer, "       SCOPE(QMGR) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       SCOPE(CELL) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

sprintf( ParmBuffer, "       QDEPTHHI(%d) +\n", DefnLQ.QDepthHighLimit );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       QDEPTHLO(%d) +\n", DefnLQ.QDepthLowLimit );
fputs( ParmBuffer, fp );

if ( DefnLQ.QDepthMaxEvent == MQEVR_ENABLED ) {
   sprintf( ParmBuffer, "       QDPMAXEV(ENABLED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       QDPMAXEV(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.QDepthHighEvent == MQEVR_ENABLED ) {
   sprintf( ParmBuffer, "       QDPHIEV(ENABLED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       QDPHIEV(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.QDepthLowEvent == MQEVR_ENABLED ) {
   sprintf( ParmBuffer, "       QDPLOEV(ENABLED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       QDPLOEV(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

sprintf( ParmBuffer, "       QSVCINT(%d) +\n", DefnLQ.QServiceInterval );
fputs( ParmBuffer, fp );

switch ( DefnLQ.QServiceIntervalEvent ) {
case MQQSIE_OK:
   sprintf( ParmBuffer, "       QSVCIEV(OK)\n" );
   fputs( ParmBuffer, fp );
   break;
case MQQSIE_NONE:
   sprintf( ParmBuffer, "       QSVCIEV(NONE)\n" );
   fputs( ParmBuffer, fp );
   break;
case MQQSIE_HIGH:
   sprintf( ParmBuffer, "       QSVCIEV(HIGH)\n" );
   fputs( ParmBuffer, fp );
   break;
```

```
   } /* endswitch */

   sprintf( ParmBuffer, "\n" );
   fputs( ParmBuffer, fp );

   fclose(fp);

}

/* ------------------------------------------------------------------------ */
/*                                                                          */
/* The queue manager returns strings of the maximum length for each         */
/* specific parameter, padded with blanks.                                  */
/*                                                                          */
/* We are interested in only the nonblank characters so will extract them   */
/* from the message buffer, and terminate the string with a null, \0.       */
/*                                                                          */
/* ------------------------------------------------------------------------ */
void MQParmCpy( char *target, char *source, int length )
{
   int   counter=0;

   while ( counter < length && source[counter] != ' ' ) {
      target[counter] = source[counter];
      counter++;
   } /* endwhile */

   if ( counter < length) {
      target[counter] = '\0';
   } /* endif */
}

MQHOBJ OpenQ( MQHCONN hConn, MQCHAR48 QName, MQLONG OpenOpts)
{
   MQHOBJ Hobj;
   MQLONG CompCode, Reason;

   ObjDesc.ObjectType = MQOT_Q;
   strncpy(ObjDesc.ObjectName, QName, MQ_Q_NAME_LENGTH);

   MQOPEN(hConn,     /* connection handle                     */
          &ObjDesc,  /* object descriptor for queue           */
          OpenOpts,  /* open options                          */
          &Hobj,     /* object handle                         */
          &CompCode, /* MQOPEN completion code                */
          &Reason);  /* reason code                           */

   /* report reason, if any; stop if failed                   */
   if (Reason != MQRC_NONE)
   {
     printf("MQOPEN for %s ended with Reason Code %d and Comp Code %d\n",
      QName,
      Reason,
      CompCode);
     exit( -1 );
   }

   return Hobj;
}

void PutMsg(MQHCONN hConn,
      MQCHAR8 MsgFormat,
      MQHOBJ hQName,
      MQCHAR48 QName,
      MQBYTE *UserMsg,
      MQLONG UserMsgLen)
{
```

```
   MQLONG CompCode, Reason;

   /* setup the message descriptor prior to putting the message */
   md.Report         = MQRO_NONE;
   md.MsgType        = MQMT_REQUEST;
   md.Expiry         = MQEI_UNLIMITED;
   md.Feedback       = MQFB_NONE;
   md.Encoding       = MQENC_NATIVE;
   md.Priority       = MQPRI_PRIORITY_AS_Q_DEF;
   md.Persistence    = MQPER_PERSISTENCE_AS_Q_DEF;
   md.MsgSeqNumber   = 1;
   md.Offset         = 0;
   md.MsgFlags       = MQMF_NONE;
   md.OriginalLength = MQOL_UNDEFINED;

   memcpy(md.GroupId,  MQGI_NONE, sizeof(md.GroupId));
   memcpy(md.Format,   MsgFormat, sizeof(md.Format) );
   memcpy(md.ReplyToQ, QName,     sizeof(md.ReplyToQ) );

   /* reset MsgId and CorrelId to get a new one                */
   memcpy(md.MsgId,    MQMI_NONE, sizeof(md.MsgId) );
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId) );

   MQPUT(hConn,             /* connection handle               */
         hQName,            /* object handle                   */
         &md,               /* message descriptor              */
         &pmo,              /* default options                 */
         UserMsgLen,        /* message length                  */
         (MQBYTE *)UserMsg, /* message buffer                  */
         &CompCode,         /* completion code                 */
         &Reason);          /* reason code                     */

   if (Reason != MQRC_NONE) {
      printf("MQPUT ended with with Reason Code %d and Comp Code %d\n",
       Reason, CompCode);
      exit( -1 );
   }
}

void GetMsg(MQHCONN hConn, MQLONG MQParm, MQHOBJ hQName,
      MQBYTE *UserMsg, MQLONG ReadBufferLen)
{
   MQLONG CompCode, Reason, msglen;

   gmo.Options      = MQParm;
   gmo.WaitInterval = 15000;

   /* reset MsgId and CorrelId to get a new one                */
   memcpy(md.MsgId,    MQMI_NONE, sizeof(md.MsgId) );
   memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId) );

   MQGET(hConn,             /* connection handle               */
         hQName,            /* object handle                   */
         &md,               /* message descriptor              */
         &gmo,              /* get message options             */
         ReadBufferLen,     /* Buffer length                   */
         (MQBYTE *)UserMsg, /* message buffer                  */
         &msglen,           /* message length                  */
         &CompCode,         /* completion code                 */
         &Reason);          /* reason code                     */

   if (Reason != MQRC_NONE) {
      printf("MQGET ended with Reason Code %d and Comp Code %d\n",
       Reason, CompCode);
      exit( -1 );
   }
}
```

WebSphere MQ Administration Interface

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

MQAI reference

Reference information for the MQAI.

A list of reference information for the MQAI.

There are two types of selector: *user selector* and *system selector*. These are described in "MQAI Selectors" on page 1331.

There are three types of call:

- Data-bag manipulation calls for configuring data bags:
  - "mqAddBag" on page 1255
  - "mqAddByteString" on page 1256
  - "mqAddByteStringFilter" on page 1258
  - "mqAddInquiry" on page 1260
  - "mqAddInteger" on page 1262
  - "mqAddInteger64" on page 1263
  - "mqAddIntegerFilter" on page 1265
  - "mqAddString" on page 1266
  - "mqAddStringFilter" on page 1268
  - "mqClearBag" on page 1273
  - "mqCountItems" on page 1274
  - "mqCreateBag" on page 1276
  - "mqDeleteBag" on page 1279
  - "mqDeleteItem" on page 1280
  - "mqInquireBag" on page 1287
  - "mqInquireByteString" on page 1290
  - "mqInquireByteStringFilter" on page 1292
  - "mqInquireInteger" on page 1295
  - "mqInquireInteger64" on page 1297
  - "mqInquireIntegerFilter" on page 1299
  - "mqInquireItemInfo" on page 1301
  - "mqInquireString" on page 1303
  - "mqInquireStringFilter" on page 1306
  - "mqSetByteString" on page 1311
  - "mqSetByteStringFilter" on page 1314
  - "mqSetInteger" on page 1316
  - "mqSetInteger64" on page 1318
  - "mqSetIntegerFilter" on page 1321
  - "mqSetString" on page 1323
  - "mqSetStringFilter" on page 1326
  - "mqTruncateBag" on page 1329
- Command calls for sending and receiving administration commands and PCF messages:
  - "mqBagToBuffer" on page 1270
  - "mqBufferToBag" on page 1272

- "mqExecute" on page 1282
- "mqGetBag" on page 1285
- "mqPutBag" on page 1310

- Utility calls for handling blank-padded and null-terminated strings:
  - "mqPad" on page 1309
  - "mqTrim" on page 1328

These calls are described in alphabetical order in the following sections.

mqAddBag:

The mqAddBag call nests a bag in another bag.

Syntax for mqAddBag

mqAddBag *(Bag, Selector, ItemValue, CompCode, Reason)*

Parameters for mqAddBag

Bag **(MQHBAG) - input**
Bag handle into which the item is to be added.

The bag must be a user bag. This means that it must have been created using the MQCBO_USER_BAG option on the mqCreateBag call. If the bag was not created in this way, MQRC_WRONG_BAG_TYPE results.

Selector **(MQLONG) - input**
Selector identifying the item to be nested.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option, the selector must be in the range MQGA_FIRST through MQGA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue **(MQHBAG) - input**
The bag which is to be nested.

If the bag is not a group bag, MQRC_BAG_WRONG_TYPE results. If an attempt is made to add a bag to itself, MQRC_HBAG_ERROR results.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddBag call:

MQRC_BAG_WRONG_TYPE
Wrong type of bag for intended use (either Bag or ItemValue).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes for mqAddBag

If a bag with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.

C language invocation for mqAddBag

```
mqAddBag (Bag, Selector, ItemValue, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQHBAG   ItemValue; /* Nested bag handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddBag

(Supported on Windows only.)

```
mqAddGroup Bag, Selector, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemValue As Long 'Nested bag handle'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

Note: The mqAddBag call can be used with user bags only; you cannot add nested bags to administration or command bags. You can only nest group bags.

mqAddByteString:

The mqAddByteString call adds a byte string identified by a user selector to the end of a specified bag.

Syntax for mqAddByteString

mqAddByteString *(Bag, Selector, BufferLength, Buffer, CompCode, Reason)*

Parameters for mqAddByteString

Bag **(MQHBAG) - input**
Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQBA_FIRST through MQBA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength **(MQLONG) - input**
The length in bytes of the string contained in the *Buffer* parameter. The value must be zero or greater.

Buffer **(MQBYTE** × *BufferLength***) - input**
Buffer containing the byte string.

The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter. In all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddByteString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddByteString

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddByteString

```
mqAddByteString (hBag, Selector, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   BufferLength;  /* Buffer length */
PMQBYTE  Buffer         /* Buffer containing item value */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddByteString

(Supported on Windows only.)

```
mqAddByteString Bag, Selector, BufferLength, Buffer, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Byte 'Buffer containing item value'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqAddByteStringFilter:

The mqAddByteStringFilter call adds a byte string filter identified by a user selector to the end of a specified bag.

Syntax for mqAddByteStringFilter

mqAddByteStringFilter *(Bag, Selector, BufferLength, Buffer, Operator, CompCode, Reason)*

Parameters for mqAddByteStringFilter

Bag **(MQHBAG) - input**
Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQBA_FIRST through MQBA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength **(MQLONG) - input**
The length in bytes of the condition byte string contained in the *Buffer* parameter. The value must be zero or greater.

Buffer **(MQBYTE** × *BufferLength***) - input**
Buffer containing the condition byte string.

The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter. In all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

Operator **(MQLONG) - input**
The byte string filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddByteStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddByteStringFilter

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddByteStringFilter

```
mqAddByteStringFilter (hBag, Selector, BufferLength, Buffer, Operator,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   BufferLength;  /* Buffer length */
PMQBYTE  Buffer         /* Buffer containing item value */
MQLONG   Operator       /* Operator */
PMQLONG  CompCode;      /* Completion code */
PMQLONG  Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddByteStringFilter

(Supported on Windows only.)

```
mqAddByteStringFilter Bag, Selector, BufferLength, Buffer, Operator, CompCode,
Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As String 'Buffer containing item value'
Dim Operator     As Long 'Operator'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqAddInquiry:

The mqAddInquiry call can be used with administration bags only; it is specifically for administration purposes.

The mqAddInquiry call adds a selector to an administration bag. The selector refers to a IBM WebSphere MQ object attribute that is to be returned by a PCF INQUIRE command. The value of the `Selector` parameter specified on this call is added to the end of the bag, as the value of a data item that has the selector value MQIACF_INQUIRY.

Syntax for mqAddInquiry

mqAddInquiry *(Bag, Selector, CompCode, Reason)*

Parameters for mqAddInquiry

Bag **(MQHBAG) - input**
Bag handle.

The bag must be an administration bag; that is, it must have been created with the MQCBO_ADMIN_BAG option on the mqCreateBag call. If the bag was not created this way, MQRC_BAG_WRONG_TYPE results.

Selector **(MQLONG) - input**
Selector of the IBM WebSphere MQ object attribute that is to be returned by the appropriate INQUIRE administration command.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInquiry call:

MQRC_BAG_WRONG_TYPE
Wrong type of bag for intended use.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddInquiry

1. When the administration message is generated, the MQAI constructs an integer list with the MQIACF_*_ATTRS or MQIACH_*_ATTRS selector that is appropriate to the `Command` value specified on the mqExecute, mqPutBag, or mqBagToBuffer call. It then adds the values of the attribute selectors specified by the mqAddInquiry call.
2. If the `Command` value specified on the mqExecute, mqPutBag, or mqBagToBuffer call is not recognized by the MQAI, MQRC_INQUIRY_COMMAND_ERROR results. Instead of using the mqAddInquiry call, this can be overcome by using the mqAddInteger call with the appropriate MQIACF_*_ATTRS or MQIACH_*_ATTRS selector and the `ItemValue` parameter of the selector being inquired.

C language invocation for mqAddInquiry

```
mqAddInquiry (Bag, Selector, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddInquiry

(Supported on Windows only.)

```
mqAddInquiry Bag, Selector, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag      As Long 'Bag handle'
Dim Selector As Long 'Selector'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

Supported INQUIRE command codes

- MQCMD_INQUIRE_AUTH_INFO
- MQCMD_INQUIRE_AUTH_RECS
- MQCMD_INQUIRE_AUTH_SERVICE
- MQCMD_INQUIRE_CF_STRUC
- MQCMD_INQUIRE_CHANNEL
- MQCMD_INQUIRE_CHANNEL_STATUS
- MQCMD_INQUIRE_CLUSTER_Q_MGR
- MQCMD_INQUIRE_CONNECTION
- MQCMD_INQUIRE_LISTENER
- MQCMD_INQUIRE_LISTENER_STATUS
- MQCMD_INQUIRE_NAMELIST
- MQCMD_INQUIRE_PROCESS
- MQCMD_INQUIRE_Q
- MQCMD_INQUIRE_Q_MGR
- MQCMD_INQUIRE_Q_MGR_STATUS
- MQCMD_INQUIRE_Q_STATUS
- MQCMD_INQUIRE_SECURITY

For an example that demonstrates the use of supported INQUIRE command codes, see Inquiring about queues and printing information (amqsailq.c).

mqAddInteger:

The mqAddInteger call adds an integer item identified by a user selector to the end of a specified bag.

Syntax for mqAddInteger

mqAddInteger *(Bag, Selector, ItemValue, CompCode, Reason)*

Parameters for mqAddInteger

Bag **(MQHBAG) - input**
Handle of the bag to be modified.

This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify identifies a system bag.

Selector **(MQLONG)**
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue **(MQLONG) - input**
The integer value to be placed in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddInteger

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily next to the existing instance.

2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddInteger

```
mqAddInteger (Bag, Selector, ItemValue, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQLONG   ItemValue; /* Integer value */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddInteger

(Supported on Windows only.)

```
mqAddInteger Bag, Selector, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqAddInteger64:

The mqAddInteger64 call adds a 64-bit integer item identified by a user selector to the end of a specified bag.

Syntax for mqAddInteger64

mqAddInteger64 *(Bag, Selector, ItemValue, CompCode, Reason)*

Parameters for mqAddInteger64

Bag **(MQHBAG) - input**
Handle of the bag to be modified.

This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify identifies a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue **(MQINT64) - input**
The 64-bit integer value to be placed in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInteger64 call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddInteger64

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddInteger64

```
mqAddInteger64 (Bag, Selector, ItemValue, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQINT64  ItemValue; /* Integer value */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddInteger64

(Supported on Windows only.)

```
mqAddInteger64 Bag, Selector, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim Item Value As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqAddIntegerFilter:

The mqAddIntegerFilter call adds an integer filter identified by a user selector to the end of a specified bag.

Syntax for mqAddIntegerFilter

mqAddIntegerFilter *(Bag, Selector, ItemValue, Operator, CompCode, Reason)*

Parameters for mqAddIntegerFilter

Bag **(MQHBAG) - input**
Handle of the bag to be modified.

This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify identifies a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue **(MQLONG) - input**
The integer condition value to be placed in the bag.

Operator **(MQLONG) - input**
The integer filter operator to be placed in the bag. Valid operators take the form MQCFOP_*.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddIntegerFilter call:

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddIntegerFilter

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation for mqAddIntegerFilter

```
mqAddIntegerFilter (Bag, Selector, ItemValue, Operator, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQLONG   ItemValue; /* Integer value */
MQLONG   Operator;  /* Item operator */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddIntegerFilter

(Supported on Windows only.)

```
mqAddIntegerFilter Bag, Selector, ItemValue, Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemValue As Long 'Integer value'
Dim Operator  As Long 'Item Operator'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqAddString:

The mqAddString call adds a character data item identified by a user selector to the end of a specified bag.

Syntax for mqAddString

mqAddString *(Bag, Selector, BufferLength, Buffer, CompCode, Reason)*

Parameters for mqAddString

Bag **(MQHBAG) - input**
Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength **(MQLONG) - input**
The length in bytes of the string contained in the *Buffer* parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED:

- If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null is not added to the bag as part of the string.
- If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present. Nulls do not delimit the string.

Buffer **(MQCHAR** × *BufferLength***) - input**
Buffer containing the character string.

The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter. In all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_CODED_CHAR_SET_ID_ERROR
Bag CCSID is MQCCSI_EMBEDDED.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddString

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.
3. The Coded Character Set ID associated with this string is copied from the current CCSID of the bag.

C language invocation for mqAddString

```
mqAddString (hBag, Selector, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer         /* Buffer containing item value */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddString

(Supported on Windows only.)

```
mqAddString Bag, Selector, BufferLength, Buffer, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As String 'Buffer containing item value'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqAddStringFilter:

The mqAddStringFilter call adds a string filter identified by a user selector to the end of a specified bag.

Syntax for mqAddStringFilter

mqAddStringFilter *(Bag, Selector, BufferLength, Buffer, Operator, CompCode, Reason)*

Parameters for mqAddStringFilter

Bag **(MQHBAG) - input**
Handle of the bag to be modified.

This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the data type of this occurrence must be the same as the data type of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength **(MQLONG) - input**
The length in bytes of the character condition string contained in the *Buffer* parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED:

- If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null is not added to the bag as part of the string.
- If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present. Nulls do not delimit the string.

Buffer **(MQCHAR** × *BufferLength***) - input**
Buffer containing the character condition string.

The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter. In all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

Operator **(MQLONG) - input**
The string filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_CODED_CHAR_SET_ID_ERROR
Bag CCSID is MQCCSI_EMBEDDED.

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of this occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqAddStringFilter

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.
3. The Coded Character Set ID associated with this string is copied from the current CCSID of the bag.

C language invocation for mqAddStringFilter

```
mqAddStringFilter (hBag, Selector, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer         /* Buffer containing item value */
MQLONG   Operator       /* Operator */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqAddStringFilter

(Supported on Windows only.)

```
mqAddStringFilter Bag, Selector, BufferLength, Buffer, Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As String 'Buffer containing item value'
Dim Operator     As Long 'Item operator'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqBagToBuffer:

The mqBagToBuffer call converts the bag into a PCF message in the supplied buffer.

Syntax for mqBagToBuffer

mqBagToBuffer *(OptionsBag, DataBag, BufferLength, Buffer, DataLength, CompCode, Reason)*

Parameters for mqBagToBuffer

OptionsBag **(MQHBAG) - input**
Handle of the bag containing options that control the processing of the call. This is a reserved parameter; the value must be MQHB_NONE.

DataBag **(MQHBAG) - input**
The handle of the bag to convert.

If the bag contains an administration message and mqAddInquiry was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command that is recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

If the bag contains nested system bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

BufferLength **(MQLONG) - input**
Length in bytes of the buffer supplied.

If the buffer is too small to accommodate the message generated, MQRC_BUFFER_LENGTH_ERROR results.

Buffer **(MQBYTE** × *BufferLength***) - output**
The buffer to hold the message.

DataLength **(MQLONG) - output**
The length in bytes of the buffer required to hold the entire bag. If the buffer is not long enough, the contents of the buffer are undefined but the DataLength is returned.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqBagToBuffer call:

MQRC_BAG_WRONG_TYPE
Input data bag is a group bag.

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid or buffer too small. (Required length returned in *DataLength*.)

MQRC_DATA_LENGTH_ERROR
DataLength parameter not valid (invalid parameter address).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInquiry used with a command code that is not recognized as an INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested system bags.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items or a supported option has an invalid value.

MQRC_PARAMETER_MISSING
An administration message requires a parameter that is not present in the bag.

Note: This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes for mqBagToBuffer

1. The PCF message is generated with an encoding of MQENC_NATIVE for the numeric data.
2. The buffer that holds the message can be null if the `BufferLength` is zero. This is useful if you use the mqBagToBuffer call to calculate the size of buffer necessary to convert your bag.

C language invocation for mqBagToBuffer

```
mqBagToBuffer (OptionsBag, DataBag, BufferLength, Buffer, &DataLength,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   OptionsBag;    /* Options bag handle */
MQHBAG   DataBag;       /* Data bag handle */
MQLONG   BufferLength;  /* Buffer length */
MQBYTE  Buffer[n];      /* Buffer to contain PCF */
MQLONG  DataLength;     /* Length of PCF returned in buffer */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqBagToBuffer

(Supported on Windows only.)

```
mqBagToBuffer OptionsBag, DataBag, BufferLength, Buffer, DataLength,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim OptionsBag   As Long 'Options bag handle'
Dim DataBag      As Long 'Data bag handle'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer to contain PCF'
Dim DataLength   As Long 'Length of PCF returned in buffer'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqBufferToBag:

The mqBufferToBag call converts the supplied buffer into bag form.

Syntax for mqBufferToBag

mqBufferToBag *(OptionsBag, BufferLength, Buffer, DataBag, CompCode, Reason)*

Parameters for mqBufferToBag

OptionsBag **(MQHBAG) - input**
Handle of the bag containing options that control the processing of the call. This is a reserved parameter; the value must be MQHB_NONE.

BufferLength **(MQLONG) - input**
Length in bytes of the buffer.

Buffer **(MQBYTE** × *BufferLength***) - input**
Pointer to the buffer containing the message to be converted.

Databag **(MQHBAG) - input/output**
Handle of the bag to receive the message. The MQAI performs an mqClearBag call on the bag before placing the message in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqBufferToBag call:

MQRC_BAG_CONVERSION_ERROR
Data could not be converted into a bag. This indicates a problem with the format of the data to be converted into a bag (for example, the message is not a valid PCF).

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of second occurrence of selector differs from data type of first occurrence.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items, or a supported option has a value that is not valid.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqBufferToBag

The buffer must contain a valid PCF message. The encoding of numeric data in the buffer must be MQENC_NATIVE.

The Coded Character Set ID of the bag is unchanged by this call.

C language invocation for mqBufferToBag

```
mqBufferToBag (OptionsBag, BufferLength, Buffer, DataBag,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   OptionsBag;    /* Options bag handle */
MQLONG   BufferLength;  /* Buffer length  */
MQBYTE   Buffer[n];     /* Buffer containing PCF */
MQHBAG   DataBag;       /* Data bag handle */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqBufferToBag

(Supported on Windows only.)

```
mqBufferToBag OptionsBag, BufferLength, Buffer, DataBag,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim OptionsBag   As Long 'Options bag handle'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer containing PCF'
Dim DataBag      As Long 'Data bag handle'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqClearBag:

The mqClearBag call deletes all user items from the bag, and resets system items to their initial values.

Syntax for mqClearBag

mqClearBag *(Bag, CompCode, Reason)*

Parameters for mqClearBag

Bag **(MQHBAG) - input**
Handle of the bag to be cleared. This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqClearBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqClearBag

1. If the bag contains system bags, they are also deleted.
2. The call cannot be used to clear system bags.

C language invocation for mqClearBag

```
mqClearBag (Bag, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqClearBag

(Supported on Windows only.)

```
mqClearBag Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag      As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqCountItems:

The mqCountItems call returns the number of occurrences of user items, system items, or both, that are stored in a bag with the same specific selector.

Syntax for mqCountItems

mqCountItems *(Bag, Selector, ItemCount, CompCode, Reason)*

Parameters for mqCountItems

Bag **(MQHBAG) - input**
Handle of the bag with items that are to be counted. This can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector of the data items to count.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI. MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the specified selector is not present in the bag, the call succeeds and zero is returned for *ItemCount*.

The following special values can be specified for *Selector*:

MQSEL_ALL_SELECTORS
All user and system items are to be counted.

MQSEL_ALL_USER_SELECTORS
All user items are to be counted; system items are excluded from the count.

MQSEL_ALL_SYSTEM_SELECTORS
All system items are to be counted; user items are excluded from the count.

ItemCount **(MQLONG) - output**
Number of items of the specified type in the bag (can be zero).

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqCountItems call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_ITEM_COUNT_ERROR
ItemCount parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

Usage notes for mqCountItems

This call counts the number of data items, not the number of unique selectors in the bag. A selector can occur multiple times, so there might be fewer unique selectors in the bag than data items.

C language invocation for mqCountItems

```
mqCountItems (Bag, Selector, &ItemCount, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG  ItemCount;      /* Number of items */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqCountItems

(Supported on Windows only.)

```
mqCountItems Bag, Selector, ItemCount, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag;      As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemCount As Long 'Number of items'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqCreateBag:

The mqCreateBag call creates a new bag.

Syntax for mqCreateBag

mqCreateBag *(Options, Bag, CompCode, Reason)*

Parameters for mqCreateBag

Options **(MQLONG) - input**
Options for creation of the bag.

The following are valid:

MQCBO_ADMIN_BAG
Specifies that the bag is for administering IBM WebSphere MQ objects. MQCBO_ADMIN_BAG automatically implies the MQCBO_LIST_FORM_ALLOWED, MQCBO_REORDER_AS_REQUIRED, and MQCBO_CHECK_SELECTORS options.

Administration bags are created with the MQIASY_TYPE system item set to MQCFT_COMMAND.

MQCBO_COMMAND_BAG
Specifies that the bag is a command bag. MQCBO_COMMAND_BAG is an alternative to the administration bag (MQCBO_ADMIN_BAG) and MQRC_OPTIONS_ERROR results if both are specified.

A command bag is processed in the same way as a user bag except that the value of the MQIASY_TYPE system item is set to MQCFT_COMMAND when the bag is created.

The command bag is also created for administering objects but they are not used to send administration messages to a command server as an administration bag is. The bag options assume the following default values:

- MQCBO_LIST_FORM_INHIBITIED
- MQCBO_DO_NOT_REORDER
- MQCBO_DO_NOT_CHECK_SELECTORS

Therefore, the MQAI does not change the order of data items or create lists within a message as with administration bags.

MQCBO_GROUP_BAG
Specifies that the bag is a group bag. This means that the bag is used to hold a set of grouped items. Group bags cannot be used for the administration of IBM WebSphere MQ objects. The bag options assume the following default values:

- MQCBO_LIST_FORM_ALLOWED
- MQCBO_REORDER_AS_REQUIRED
- MQCBO_DO_NOT_CHECK_SELECTORS

Therefore, the MQAI can change the order of data items or create lists within a bag of grouped items.

Group bags are created with two system selectors: MQIASY_BAG_OPTIONS and MQIASY_CODED_CHAR_SET_ID.

If a group bag is nested in a bag in which MQCBO_CHECK_SELECTORS was specified, the group bag to be nested has its selectors checked at that point whether MQCBO_CHECK_SELECTORS was specified when the group bag was created.

MQCBO_USER_BAG
Specifies that the bag is a user bag. MQCBO_USER_BAG is the default bag-type option. User

bags can also be used for the administration of IBM WebSphere MQ objects, but the MQCBO_LIST_FORM_ALLOWED and MQCBO_REORDER_AS_REQUIRED options must be specified to ensure correct generation of the administration messages.

User bags are created with the MQIASY_TYPE system item set to MQCFT_USER.

For user bags, one or more of the following options can be specified:

MQCBO_LIST_FORM_ALLOWED
Specifies that the MQAI can use the more compact list form in the message sent whenever there are two or more adjacent occurrences of the same selector in the bag. However, the items cannot be reordered if this option is used. Therefore, if the occurrences of the selector are not adjacent in the bag, and MQCBO_REORDER_AS_REQUIRED is not specified, the MQAI cannot use the list form for that particular selector.

If the data items are character strings, these strings must have the same Character Set ID and the same selector, in order to be compacted into list form. If the list form is used, the shorter strings are padded with blanks to the length of the longest string.

This option must be specified if the message to be sent is an administration message but MQCBO_ADMIN_BAG is not specified.

Note: MQCBO_LIST_FORM_ALLOWED does not imply that the MQAI definitely uses the list form. The MQAI considers various factors in deciding whether to use the list form.

MQCBO_LIST_FORM_INHIBITED
Specifies that the MQAI cannot use the list form in the message sent, even if there are adjacent occurrences of the same selector in the bag. MQCBO_LIST_FORM_INHIBITED is the default list-form option.

MQCBO_REORDER_AS_REQUIRED
Specifies that the MQAI can change the order of the data items in the message sent. This option does not affect the order of the items in the sending bag.

This option means that you can insert items into a data bag in any order. That is, the items do not need to be inserted in the way that they must be in the PCF message, because the MQAI can reorder these items as required.

If the message is a user message, the order of the items in the receiving bag is the same as the order of the items in the message. This order can be different from the order of the items in the sending bag.

If the message is an administration message, the order of the items in the receiving bag is determined by the message received.

This option must be specified if the message to be sent is an administration message but MQCBO_ADMIN is not specified.

MQCBO_DO_NOT_REORDER
Specifies that the MQAI cannot change the order of data items in the message sent. Both the message sent and the receiving bag contain the items in the same order as they occur in the sending bag. This option is the default ordering option.

MQCBO_CHECK_SELECTORS
Specifies that user selectors (selectors that are zero or greater) must be checked to ensure that the selector is consistent with the data type implied by the mqAddInteger, mqAddInteger64, mqAddIntegerFilter, mqAddString, mqAddStringFilter, mqAddByteString, mqAddByteStringFilter, mqSetInteger, mqSetInteger64, mqSetIntegerFilter, mqSetString, mqSetStringFilter, mqSetByteString, or mqSetByteStringFilter call:

- For the integer, 64-bit integer, and integer filter calls, the selector must be in the range MQIA_FIRST through MQIA_LAST.
- For the string and string filter calls, the selector must be in the range MQCA_FIRST through MQCA_LAST.
- For byte string and byte string filter calls, the selector must be in the range MQBA_FIRST through MQBA_LAST
- For group bag calls, the selector must be in the range MQGA_FIRST through MQGA_LAST
- For the handle calls, the selector must be in the range MQHA_FIRST through MQHA_LAST.

The call fails if the selector is outside the valid range. System selectors (selectors less than zero) are always checked, and if a system selector is specified, it must be one that is supported by the MQAI.

MQCBO_DO_NOT_CHECK_SELECTORS
Specifies that user selectors (selectors that are zero or greater) are not checked. Any selector that is zero or positive can be used with any call. This option is the default selectors option. System selectors (selectors less than zero) are always checked.

MQCBO_NONE
Specifies that all options must have their default values. This option is provided to aid program documentation, and must not be specified with any of the options that have a nonzero value.

The following list summarizes the default option values:

- MQCBO_USER_BAG
  - MQCBO_LIST_FORM_INHIBITIED
  - MQCBO_DO_NOT_REORDER
  - MQCBO_DO_NOT_CHECK_SELECTORS

Bag **(MQHBAG) - output**
The handle of the bag created by the call.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqCreateBag call:

MQRC_HBAG_ERROR
Bag handle not valid (invalid parameter address or the parameter location is read-only).

MQRC_OPTIONS_ERROR
Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes for mqCreateBag

Any options used for creating your bag are contained in a system item within the bag when it is created.

C language invocation for mqCreateBag

```
mqCreateBag (Options, &Bag, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQLONG   Options;       /* Bag options */
MQHBAG  Bag;            /* Bag handle */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqCreateBag

(Supported on Windows only.)

```
mqCreateBag Options, Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Options  As Long 'Bag options'
Dim Bag      As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqDeleteBag:

The mqDeleteBag call deletes the specified bag.

Syntax for mqDeleteBag

mqDeleteBag *(Bag, CompCode, Reason)*

Parameters for mqDeleteBag

Bag **(MQHBAG) - input/output**
The handle of the bag to be deleted. This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_DELETABLE results if you specify the handle of a system bag. The handle is reset to MQHB_UNUSABLE_HBAG.

If the bag contains system-generated bags, they are also deleted.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqDeleteBag call:

MQRC_HBAG_ERROR
Bag handle not valid, or invalid parameter address, or parameter location is read only.

MQRC_SYSTEM_BAG_NOT_DELETABLE
System bag cannot be deleted.

Usage notes for mqDeleteBag

1. Delete any bags created with mqCreateBag.
2. Nested bags are deleted automatically when the containing bag is deleted.

C language invocation for mqDeleteBag

```
mqDeleteBag (&Bag, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqDeleteBag

(Supported on Windows only.)

```
mqDeleteBag Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag;     As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqDeleteItem:

The mqDeleteItem call removes one or more user items from a bag.

Syntax for mqDeleteItem

mqDeleteItem *(Bag, Selector, ItemIndex, CompCode, Reason)*

Parameters for mqDeleteItem

Hbag **(MQHBAG) - input**
Handle of the bag to be modified.

This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if it is a system bag.

Selector **(MQLONG) - input**
Selector identifying the user item to be deleted.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

The following special values are valid:

MQSEL_ANY_SELECTOR
The item to be deleted is a user item identified by the `ItemIndex` parameter, the index relative to the set of items that contains both user and system items.

MQSEL_ANY_USER_SELECTOR
The item to be deleted is a user item identified by the `ItemIndex` parameter, the index relative to the set of user items.

If an explicit selector value is specified, but the selector is not present in the bag, the call succeeds if MQIND_ALL is specified for `ItemIndex`, and fails with reason code MQRC_SELECTOR_NOT_PRESENT if MQIND_ALL is not specified.

ItemIndex **(MQLONG) - input**
Index of the data item to be deleted.

The value must be zero or greater, or one of the following special values:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results. If MQIND_NONE is specified with one of the MQSEL_XXX_SELECTOR values, MQRC_INDEX_ERROR results.

MQIND_ALL
This specifies that all occurrences of the selector in the bag are to be deleted. If MQIND_ALL is specified with one of the MQSEL_XXX_SELECTOR values, MQRC_INDEX_ERROR results. If MQIND_ALL is specified when the selector is not present within the bag, the call succeeds.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and

must be zero or greater. If ItemIndex identifies a system selector MQRC_SYSTEM_ITEM_NOT_DELETABLE results. If MQSEL_ANY_USER_SELECTOR is specified for the Selector parameter, the ItemIndex parameter is the index relative to the set of user items, and must be zero or greater.

If an explicit selector value is specified, ItemIndex is the index relative to the set of items that have that selector value, and can be MQIND_NONE, MQIND_ALL, zero, or greater.

If an explicit index is specified (that is, not MQIND_NONE or MQIND_ALL) and the item is not present in the bag, MQRC_INDEX_NOT_PRESENT results.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqDeleteItem call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
MQIND_NONE or MQIND_ALL specified with one of the MQSEL_ANY_XXX_SELECTOR values.

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag is read only and cannot be altered.

MQRC_SYSTEM_ITEM_NOT_DELETABLE
System item is read only and cannot be deleted.

Usage notes for mqDeleteItem

1. Either a single occurrence of the specified selector can be removed, or all occurrences of the specified selector.
2. The call cannot remove system items from the bag, or remove items from a system bag. However, the call can remove the handle of a system bag from a user bag. This way, a system bag can be deleted.

C language invocation for mqDeleteItem

```
mqDeleteItem (Bag, Selector, ItemIndex, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Hbag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Index of the data item */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqDeleteItem

(Supported on Windows only.)

```
mqDeleteItem Bag, Selector, ItemIndex, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Index of the data item'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqExecute:

The mqExecute call sends an administration command message and waits for the reply (if expected).

Syntax for mqExecute

mqExecute *(Hconn, Command, OptionsBag, AdminBag, ResponseBag, AdminQ, ResponseQ, CompCode, Reason)*

Parameters for mqExecute

Hconn **(MQHCONN) - input**
MQI Connection handle.

This is returned by a preceding MQCONN call issued by the application.

Command **(MQLONG) - input**
The command to be executed.

This should be one of the MQCMD_* values. If it is a value that is not recognized by the MQAI servicing the mqExecute call, the value is still accepted. However, if mqAddInquiry was used to insert values in the bag, the Command parameter must be an INQUIRE command recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

OptionsBag **(MQHBAG) - input**
Handle of a bag containing options that affect the operation of the call.

This must be the handle returned by a preceding mqCreateBag call or the following special value:

MQHB_NONE
No options bag; all options assume their default values.

Only the options listed in this topic can be present in the options bag (MQRC_OPTIONS_ERROR results if other data items are present).

The appropriate default value is used for each option that is not present in the bag. The following option can be specified:

MQIACF_WAIT_INTERVAL
This data item specifies the maximum time in milliseconds that the MQAI should wait for each reply message. The time interval must be zero or greater, or the special value MQWI_UNLIMITED; the default is thirty seconds. The mqExecute call completes either when all of the reply messages are received or when the specified wait interval expires without the expected reply message having been received.

Note: The time interval is an approximate quantity.

If the MQIACF_WAIT_INTERVAL data item has the wrong data type, or there is more than one occurrence of that selector in the options bag, or the value of the data item is not valid, MQRC_WAIT_INTERVAL_ERROR results.

AdminBag **(MQHBAG) - input**
Handle of the bag containing details of the administration command to be issued.

All user items placed in the bag are inserted into the administration message that is sent. It is the application's responsibility to ensure that only valid parameters for the command are placed in the bag.

If the value of the MQIASY_TYPE data item in the command bag is not MQCFT_COMMAND, MQRC_COMMAND_TYPE_ERROR results. If the bag contains nested system bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

ResponseBag **(MQHBAG) - input**
Handle of the bag where reply messages are placed.

The MQAI performs an mqClearBag call on the bag before placing reply messages in the bag. To retrieve the reply messages, the selector, MQIACF_CONVERT_RESPONSE, can be specified.

Each reply message is placed into a separate system bag, with a handle that is then placed in the response bag. Use the mqInquireBag call with selector MQHA_BAG_HANDLE to determine the handles of the system bags within the reply bag, and those bags can then be inquired to determine their contents.

If some but not all of the expected reply messages are received, MQCC_WARNING with MQRC_NO_MSG_AVAILABLE results. If none of the expected reply messages is received, MQCC_FAILED with MQRC_NO_MSG_AVAILABLE results.

Group bags cannot be used as response bags.

AdminQ **(MQHOBJ) - input**
Object handle of the queue on which the administration message is to be placed.

This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for output.

The following special value can be specified:

MQHO_NONE
This indicates that the administration message should be placed on the SYSTEM.ADMIN.COMMAND.QUEUE belonging to the currently connected queue manager. If MQHO_NONE is specified, the application need not use MQOPEN to open the queue.

ResponseQ
Object handle of the queue on which reply messages are placed.

This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for input and for inquiry.

The following special value can be specified:

MQHO_NONE
This indicates that the reply messages should be placed on a dynamic queue created automatically by the MQAI. The queue is created by opening SYSTEM.DEFAULT.MODEL.QUEUE, that must therefore have suitable characteristics. The queue created exists for the duration of the call only, and is deleted by the MQAI on exit from the mqExecute call.

CompCode
Completion code.

Reason
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqExecute call:

MQRC_*
Anything from the MQINQ, MQPUT, MQGET, or MQOPEN calls.

MQRC_BAG_WRONG_TYPE
Input data bag is a group bag.

MQRC_CMD_SERVER_NOT_AVAILABLE
The command server that processes administration commands is not available.

MQRC_COMMAND_TYPE_ERROR
The value of the MQIASY_TYPE data item in the request bag is not MQCFT_COMMAND.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInteger call used with a command code that is not a recognized INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested system bags.

MQRC_NO_MSG_AVAILABLE
Some reply messages received, but not all. Reply bag contains system-generated bags for messages that were received.

MQRC_NO_MSG_AVAILABLE
No reply messages received during the specified wait interval.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items, or a supported option has a value which is not valid.

MQRC_PARAMETER_MISSING
Administration message requires a parameter which is not present in the bag. This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_NOT_UNIQUE
Two or more instances of a selector exist within the bag for a mandatory parameter that permits one instance only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRCCF_COMMAND_FAILED
Command failed; details of failure are contained in system-generated bags within the reply bag.

Usage notes for mqExecute

1. If no *AdminQ* is specified, the MQAI checks to see if the command server is active before sending the administration command message. However, if the command server is not active, the MQAI does not start it. If you are sending many administration command messages, you are recommended to open the SYSTEM.ADMIN.COMMAND.QUEUE yourself and pass the handle of the administration queue on each administration request.
2. Specifying the MQHO_NONE value in the *ResponseQ* parameter simplifies the use of the mqExecute call, but if mqExecute is issued repeatedly by the application (for example, from within a loop), the response queue will be created and deleted repeatedly. In this situation, it is better for the application itself to open the response queue before any mqExecute call, and close it after all mqExecute calls have been issued.

3. If the administration command results in a message being sent with a message type of MQMT_REQUEST, the call waits for the time given by the MQIACF_WAIT_INTERVAL data item in the options bag.
4. If an error occurs during the processing of the call, the response bag might contain some data from the reply message, but the data will typically be incomplete.

C language invocation for mqExecute

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag,
AdminQ, ResponseQ, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* MQI connection handle */
MQLONG   Command;       /* Command to be executed */
MQHBAG   OptionsBag;    /* Handle of a bag containing options */
MQHBAG   AdminBag;      /* Handle of administration bag containing
                        /* details of administration command */
MQHBAG   ResponseBag;   /* Handle of bag for response messages */
MQHOBJ   AdminQ         /* Handle of administration queue for
                           administration messages */
MQHOBJ   ResponseQ;     /* Handle of response queue for response
                           messages */
MQLONG   pCompCode;     /* Completion code */
MQLONG   pReason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqExecute

(Supported on Windows only.)

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag,
AdminQ, ResponseQ, CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn       As Long 'MQI connection handle'
Dim Command     As Long 'Command to be executed'
Dim OptionsBag  As Long 'Handle of a bag containing options'
Dim AdminBag  As Long 'Handle of command bag containing details of
                        administration command'
Dim ResponseBag As Long 'Handle of bag for reply messages'
Dim AdminQ    As Long 'Handle of command queue for
                        administration messages'
Dim ResponseQ   As Long 'Handle of response queue for reply messages'
Dim CompCode    As Long 'Completion code'
Dim Reason      As Long 'Reason code qualifying CompCode'
```

mqGetBag:

The mqGetBag call removes a message from the specified queue and converts the message data into a data bag.

Syntax for mqGetBag

mqGetBag *(Hconn, Hobj, MsgDesc, GetMsgOpts, Bag, CompCode, Reason)*

Parameters for mqGetBag

Hconn **(MQHCONN) - input**
MQI connection handle.

Hobj **(MQHOBJ) - input**
Object handle of the queue from which the message is to be retrieved. This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for input.

MsgDesc **(MQMD) - input/output**
Message descriptor (for more information, see MQMD - Message descriptor).

If the *Format* field in the message has a value other than MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF, MQRC_FORMAT_NOT_SUPPORTED results.

If, on entry to the call, the *Encoding* field in the application's MQMD has a value other than MQENC_NATIVE and MQGMO_CONVERT is specified, MQRC_ENCODING_NOT_SUPPORTED results. Also, if MQGMO_CONVERT is not specified, the value of the *Encoding* parameter must be the retrieving application's MQENC_NATIVE; if not, again MQRC_ENCODING_NOT_SUPPORTED results.

GetMsgOpts **(MQGMO) - input/output**
Get-message options (for more information, see MQGMO - Get-message options).

MQGMO_ACCEPT_TRUNCATED_MSG cannot be specified; MQRC_OPTIONS_ERROR results if it is. MQGMO_LOCK and MQGMO_UNLOCK are not supported in a 16-bit or 32-bit Window environment. MQGMO_SET_SIGNAL is supported in a 32-bit Window environment only.

Bag **(MQHBAG) - input/output**
Handle of a bag into which the retrieved message is placed. The MQAI performs an mqClearBag call on the bag before placing the message in the bag.

MQHB_NONE
Gets the retrieved message. This provides a means of deleting messages from the queue.

If an option of MQGMO_BROWSE_* is specified, this value sets the browse cursor to the selected message; it is not deleted in this case.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating warning and error conditions can be returned from the mqGetBag call:

MQRC_*
Anything from the MQGET call or bag manipulation.

MQRC_BAG_CONVERSION_ERROR
Data could not be converted into a bag.

This indicates a problem with the format of the data to be converted into a bag (for example, the message is not a valid PCF).

If the message was retrieved destructively from the queue (that is, not browsing the queue), this reason code indicates that it has been discarded.

MQRC_BAG_WRONG_TYPE
Input data bag is a group bag.

MQRC_ENCODING_NOT_SUPPORTED
Encoding not supported; the value in the *Encoding* field of the MQMD must be MQENC_NATIVE.

MQRC_FORMAT_NOT_SUPPORTED
Format not supported; the *Format* name in the message is not MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF. If the message was retrieved destructively from the queue (that is, not browsing the queue), this reason code indicates that it has been discarded.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Data type of second occurrence of selector differs from data type of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqGetBag

1. Only messages that have a supported format can be returned by this call. If the message has a format that is not supported, the message is discarded, and the call completes with an appropriate reason code.
2. If the message is retrieved within a unit of work (that is, with the MQGMO_SYNCPOINT option), and the message has an unsupported format, the unit of work can be backed out, reinstating the message on the queue. This allows the message to be retrieved by using the MQGET call in place of the mqGetBag call.

C language invocation for mqGetBag

```
mqGetBag (hConn, hObj, &MsgDesc, &GetMsgOpts, hBag, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHCONN  hConn;         /* MQI connection handle */
MQHOBJ   hObj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQGMO    GetMsgOpts;    /* Get-message options */
MQHBAG   hBag;          /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqGetBag

(Supported on Windows only.)

```
mqGetBag (HConn, HObj, MsgDesc, GetMsgOpts, Bag, CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn      As Long 'MQI connection handle'
Dim HObj       As Long 'Object handle'
Dim MsgDesc    As Long 'Message descriptor'
Dim GetMsgOpts As Long 'Get-message options'
Dim Bag        As Long 'Bag handle'
Dim CompCode   As Long 'Completion code'
Dim Reason     As Long 'Reason code qualifying CompCode'
```

mqInquireBag:

The mqInquireBag call inquires the value of a bag handle that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireBag

mqInquireBag *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqInquireBag

Bag **(MQHBAG) - input**
Bag handle to be inquired. The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to be inquired.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for `Selector`:

MQSEL_ANY_SELECTOR
The item to be inquired is a user or system item identified by the `ItemIndex` parameter.

MQSEL_ANY_USER_SELECTOR
The item to be inquired is a user item identified by the `ItemIndex` parameter.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired is a system item identified by the `ItemIndex` parameter.

ItemIndex **(MQLONG) - input**
Index of the data item to be inquired.

The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results.

The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, the `ItemIndex` parameter is the index relative to the set of items that have that selector value and can be MQIND_NONE, zero, or greater.

ItemValue **(MQHBAG) - output**
Value of the item in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
The `ItemValue` parameter is not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present within the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireBag

```
mqInquireBag (Bag, Selector, ItemIndex, &ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Index of the data item to be inquired */
MQHBAG   ItemValue;     /* Value of item in the bag */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireBag

(Supported on Windows only.)

```
mqInquireBag (Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Index of the data item to be inquired'
Dim ItemValue As Long 'Value of item in the bag'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireByteString:

The mqInquireByteString call requests the value of a byte string data item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireByteString

mqInquireByteString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, ByteStringLength, CompCode, Reason)*

Parameters for mqInquireByteString

Bag **(MQHBAG) - input**
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex **(MQLONG) - input**
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength **(MQLONG) - input**
Length in bytes of the buffer to receive the byte string. Zero is a valid value.

Buffer **(MQBYTE × *BufferLength*) - output**
Buffer to receive the byte string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

The string is padded with nulls to the length of the buffer. If the string is longer than the buffer, the string is truncated to fit; in this case *ByteStringLength* indicates the size of the buffer needed to accommodate the string without truncation.

ByteStringLength **(MQLONG) - output**
The length in bytes of the string contained in the bag. If the *Buffer* parameter is too small, the length of the string returned is less than *ByteStringLength*.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireByteString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
ByteStringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation for mqInquireByteString

```
mqInquireByteString (Bag, Selector, ItemIndex,
BufferLength, Buffer, &StringLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQBYTE  Buffer;        /* Buffer to contain string */
MQLONG   ByteStringLength;  /* Length of byte string returned */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireByteString

(Supported on Windows only.)

```
mqInquireByteString Bag, Selector, ItemIndex,
BufferLength, Buffer, StringLength, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As Byte 'Buffer to contain string'
Dim ByteStringLength As Long   'Length of byte string returned'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqInquireByteStringFilter:

The mqInquireByteStringFilter call requests the value and operator of a byte string filter item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireByteStringFilter

mqInquireByteStringFilter *(Bag, Selector, ItemIndex, Bufferlength, Buffer, ByteStringLength, Operator, CompCode, Reason)*

Parameters for mqInquireByteStringFilter

Bag **(MQHBAG) - input**
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex **(MQLONG) - input**
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength **(MQLONG) - input**
Length in bytes of the buffer to receive the condition byte string. Zero is a valid value.

Buffer **(MQBYTE** × *BufferLength***) - output**
Buffer to receive the condition byte string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

The string is padded with blanks to the length of the buffer; the string is not null-terminated. If the string is longer than the buffer, the string is truncated to fit; in this case *ByteStringLength* indicates the size of the buffer needed to accommodate the string without truncation.

ByteStringLength **(MQLONG) - output**
The length in bytes of the condition string contained in the bag. If the *Buffer* parameter is too small, the length of the string returned is less than *StringLength*.

Operator **(MQLONG) - output**
Byte string filter operator in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireByteStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
ByteStringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation for mqInquireByteStringFilter

```
mqInquireByteStringFilter (Bag, Selector, ItemIndex,
BufferLength, Buffer, &ByteStringLength, &Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQBYTE  Buffer;        /* Buffer to contain string */
MQLONG   ByteStringLength;  /* Length of string returned */
MQLONG   Operator       /* Item operator */
PMQLONG  CompCode;      /* Completion code */
PMQLONG  Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireByteStringFilter

(Supported on Windows only.)

```
mqInquireByteStringFilter Bag, Selector, ItemIndex,
BufferLength, Buffer, ByteStringLength,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As String 'Buffer to contain string'
```

```
Dim ByteStringLength  As Long   'Length of byte string returned'
Dim Operator        As Long   'Operator'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqInquireInteger:

The mqInquireInteger call requests the value of an integer data item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireInteger

mqInquireInteger *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqInquireInteger

Bag **(MQHBAG) - input**
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to which the inquiry relates.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex **(MQLONG) - input**
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and is not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

ItemValue **(MQLONG) - output**
The value of the item in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
ItemValue parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireInteger

```
mqInquireInteger (Bag, Selector, ItemIndex, &ItemValue,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Item value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireInteger

(Supported on Windows only.)

```
mqInquireInteger Bag, Selector, ItemIndex, ItemValue,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Item value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireInteger64:

The mqInquireInteger64 call requests the value of a 64-bit integer data item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireInteger64

mqInquireInteger64 *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqInquireInteger64

Bag **(MQHBAG) - input**
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to which the inquiry relates.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex **(MQLONG) - input**
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and is not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

ItemValue **(MQINT64) - output**
The value of the item in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireInteger64 call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
ItemValue parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireInteger64

```
mqInquireInteger64 (Bag, Selector, ItemIndex, &ItemValue,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQINT64  ItemValue;     /* Item value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireInteger64

(Supported on Windows only.)

```
mqInquireInteger64 Bag, Selector, ItemIndex, ItemValue,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Item value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireIntegerFilter:

The mqInquireIntegerFilter call requests the value and operator of an integer filter item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireIntegerFilter

mqInquireIntegerFilter *(Bag, Selector, ItemIndex, ItemValue, Operator, CompCode, Reason)*

Parameters for mqInquireIntegerFilter

Bag **(MQHBAG) - input**
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to which the inquiry relates.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex **(MQLONG) - input**
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and is not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

ItemValue **(MQLONG) - output**
The condition value.

Operator **(MQLONG) - output**
Integer filter operator in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireIntegerFilter call:

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
ItemValue parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireIntegerFilter

```
mqInquireIntegerFilter (Bag, Selector, ItemIndex, &ItemValue,
&Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Item value */
MQLONG   Operator;      /* Item operator */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireIntegerFilter

(Supported on Windows only.)

```
mqInquireIntegerFilter Bag, Selector, ItemIndex, ItemValue,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Item value'
Dim Operator  As Long 'Item operator'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireItemInfo:

The mqInquireItemInfo call returns information about a specified item in a bag. The data item can be a user item or a system item.

Syntax for mqInquireItemInfo

mqInquireItemInfo *(Bag, Selector, ItemIndex, ItemType, OutSelector, CompCode, Reason)*

Parameters for mqInquireItemInfo

Bag **(MQHBAG) - input**
Handle of the bag to be inquired.

The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector identifying the item to be inquired.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The following special values can be specified for `Selector`:

MQSEL_ANY_SELECTOR
The item to be inquired is a user or system item identified by the `ItemIndex` parameter.

MQSEL_ANY_USER_SELECTOR
The item to be inquired is a user item identified by the `ItemIndex` parameter.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired is a system item identified by the `ItemIndex` parameter.

ItemIndex **(MQLONG) - input**
Index of the data item to be inquired.

The item must be present within the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The value must be zero or greater, or the following special value:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater. If an explicit selector value is specified, the `ItemIndex` parameter is the index relative to the set of items that have that selector value and can be MQIND_NONE, zero, or greater.

ItemType **(MQLONG) - output**
The data type of the specified data item.

The following can be returned:

MQITEM_BAG
Bag handle item.

MQITEM_BYTE_STRING
Byte string.

MQITEM_INTEGER
Integer item.

MQITEM_INTEGER_FILTER
Integer filter.

MQITEM_INTEGER64
64-bit integer item.

MQITEM_STRING
Character-string item.

MQITEM_STRING_FILTER
String filter.

OutSelector **(MQLONG) - output**
Selector of the specified data item.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireItemInfo call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
MQIND_NONE specified with one of the MQSEL_ANY_XXX_SELECTOR values.

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_TYPE_ERROR
`ItemType` parameter not valid (invalid parameter address).

MQRC_OUT_SELECTOR_ERROR
`OutSelector` parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqInquireItemInfo

```
mqInquireItemInfo (Bag, Selector, ItemIndex, &OutSelector, &ItemType,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector identifying item */
MQLONG   ItemIndex;     /* Index of data item */
MQLONG   OutSelector;   /* Selector of specified data item */
MQLONG   ItemType;      /* Data type of data item */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireItemInfo

(Supported on Windows only.)

```
mqInquireItemInfo Bag, Selector, ItemIndex, OutSelector, ItemType,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag         As Long 'Bag handle'
Dim Selector    As Long 'Selector identifying item'
Dim ItemIndex   As Long 'Index of data item'
Dim OutSelector As Long 'Selector of specified data item'
Dim ItemType    As Long 'Data type of data item'
Dim CompCode    As Long 'Completion code'
Dim Reason      As Long 'Reason code qualifying CompCode'
```

mqInquireString:

The mqInquireString call requests the value of a character data item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireString

mqInquireString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, StringLength, CodedCharSetId, CompCode, Reason)*

Parameters for mqInquireString

Bag **(MQHBAG) - input**
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex **(MQLONG) - input**
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength **(MQLONG) - input**
Length in bytes of the buffer to receive the string. Zero is a valid value.

Buffer **(MQCHAR** × *BufferLength***) - output**
Buffer to receive the character string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

The string is padded with blanks to the length of the buffer; the string is not null-terminated. If the string is longer than the buffer, the string is truncated to fit; in this case *StringLength* indicates the size of the buffer needed to accommodate the string without truncation.

StringLength **(MQLONG) - output**
The length in bytes of the string contained in the bag. If the *Buffer* parameter is too small, the length of the string returned is less than *StringLength*.

CodedCharSetId **(MQLONG) - output**
The coded character set identifier for the character data in the string. This parameter can be set to a null pointer if not required.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
StringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation for mqInquireString

```
mqInquireString (Bag, Selector, ItemIndex,
BufferLength, Buffer, &StringLength, &CodedCharSetId,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer;        /* Buffer to contain string */
MQLONG   StringLength;  /* Length of string returned */
MQLONG   CodedCharSetId /* Coded Character Set ID */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireString

(Supported on Windows only.)

```
mqInquireString Bag, Selector, ItemIndex,
BufferLength, Buffer, StringLength, CodedCharSetId,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As String 'Buffer to contain string'
Dim StringLength    As Long   'Length of string returned'
Dim CodedCharSetId  As Long   'Coded Character Set ID'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqInquireStringFilter:

The mqInquireStringFilter call requests the value and operator of a string filter item that is present in the bag. The data item can be a user item or a system item.

Syntax for mqInquireStringFilter

mqInquireStringFilter *(Bag, Selector, ItemIndex, Bufferlength, Buffer, StringLength, CodedCharSetId, Operator, CompCode, Reason)*

Parameters for mqInquireStringFilter

Bag **(MQHBAG) - input**
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector **(MQLONG) - input**
Selector of the item to which the inquiry relates.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex **(MQLONG) - input**
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength **(MQLONG) - input**
Length in bytes of the buffer to receive the condition string. Zero is a valid value.

Buffer **(MQCHAR** × *BufferLength***) - output**
Buffer to receive the character condition string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

The string is padded with blanks to the length of the buffer; the string is not null-terminated. If the string is longer than the buffer, the string is truncated to fit; in this case *StringLength* indicates the size of the buffer needed to accommodate the string without truncation.

StringLength **(MQLONG) - output**
The length in bytes of the condition string contained in the bag. If the *Buffer* parameter is too small, the length of the string returned is less than *StringLength*.

CodedCharSetId **(MQLONG) - output**
The coded character set identifier for the character data in the string. This parameter can be set to a null pointer if not required.

Operator **(MQLONG) - output**
String filter operator in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
StringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation for mqInquireStringFilter

```
mqInquireStringFilter (Bag, Selector, ItemIndex,
BufferLength, Buffer, &StringLength, &CodedCharSetId,
&Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer;        /* Buffer to contain string */
MQLONG   StringLength;  /* Length of string returned */
MQLONG   CodedCharSetId /* Coded Character Set ID */
MQLONG   Operator       /* Item operator */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqInquireStringFilter

(Supported on Windows only.)

```
mqInquireStringFilter Bag, Selector, ItemIndex,
BufferLength, Buffer, StringLength, CodedCharSetId,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As String 'Buffer to contain string'
Dim StringLength    As Long   'Length of string returned'
Dim CodedCharSetId  As Long   'Coded Character Set ID'
Dim Operator        As Long   'Item operator'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqPad:

The mqPad call pads a null-terminated string with blanks.

Syntax for mqPad

mqPad *(String, BufferLength, Buffer, CompCode, Reason)*

Parameters for mqPad

String **(PMQCHAR) - input**
Null-terminated string. The null pointer is valid for the address of the *String* parameter, and denotes a string of zero length.

BufferLength **(MQLONG) - input**
Length in bytes of the buffer to receive the string padded with blanks. Must be zero or greater.

Buffer **(MQCHAR** × *BufferLength***) - output**
Buffer to receive the blank-padded string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

If the number of characters preceding the first null in the *String* parameter is greater than the *BufferLength* parameter, the excess characters are omitted and MQRC_DATA_TRUNCATED results.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqPad call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_STRING_ERROR
String parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

Usage notes for mqPad

1. If the buffer pointers are the same, the padding is done in place. If not, at most *BufferLength* characters are copied into the second buffer; any space remaining, including the null-termination character, is overwritten with spaces.
2. If the *String* and *Buffer* parameters partially overlap, the result is undefined.

C language invocation for mqPad

```
mqPad (String, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR   String;          /* String to be padded */
MQLONG   BufferLength;    /* Buffer length */
PMQCHAR  Buffer           /* Buffer to contain padded string */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

Note: This call is not supported in Visual Basic.

mqPutBag:

The mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue. The contents of the bag are unchanged after the call.

Syntax for mqPutBag

mqPutBag *(Hconn, Hobj, MsgDesc, PutMsgOpts, Bag, CompCode, Reason)*

Parameters for mqPutBag

Hconn **(MQHCONN) - input**
MQI connection handle.

Hobj **(MQHOBJ) - input**
Object handle of the queue on which the message is to be placed. This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for output.

MsgDesc **(MQMD) - input/output**
Message descriptor. (For more information, see MQMD - Message descriptor.)

If the *Format* field has a value other than MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF, MQRC_FORMAT_NOT_SUPPORTED results.

If the *Encoding* field has a value other than MQENC_NATIVE, MQRC_ENCODING_NOT_SUPPORTED results.

PutMsgOpts **(MQPMO) - input/output**
Put-message options. (For more information, see MQPMO - Put-message options.)

Bag **(MQHBAG) - input**
Handle of the data bag to be converted to a message.

If the bag contains an administration message, and mqAddInquiry was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

If the bag contains nested system bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*. The following reason codes indicating error and warning conditions can be returned from the mqPutBag call:

MQRC_*
Anything from the MQPUT call or bag manipulation.

MQRC_BAG_WRONG_TYPE
Input data bag is a group bag.

MQRC_ENCODING_NOT_SUPPORTED
Encoding not supported (value in *Encoding* field in MQMD must be MQENC_NATIVE).

MQRC_FORMAT_NOT_SUPPORTED
Format not supported (name in *Format* field in MQMD must be MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInquiry call used with a command code that is not a recognized INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested system bags.

MQRC_PARAMETER_MISSING
Administration message requires a parameter that is not present in the bag. This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation for mqPutBag

```
mqPutBag (HConn, HObj, &MsgDesc, &PutMsgOpts, Bag,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  HConn;         /* MQI connection handle */
MQHOBJ   HObj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Put-message options */
MQHBAG   Bag;           /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqPutBag

(Supported on Windows only.)

```
mqPutBag (HConn, HObj, MsgDesc, PutMsgOpts, Bag,
CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn      As Long  'MQI connection handle'
Dim HObj       As Long  'Object handle'
Dim MsgDesc    As MQMD  'Message descriptor'
Dim PutMsgOpts As MQPMO 'Put-message options'
Dim Bag        As Long  'Bag handle'
Dim CompCode   As Long  'Completion code'
Dim Reason     As Long  'Reason code qualifying CompCode'
```

mqSetByteString:

The mqSetByteString call either modifies a byte string data item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax for mqSetByteString

mqSetByteString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, CompCode, Reason)*

Parameters for mqSetByteString

Bag **(MQHBAG) - input**
Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector **(MQLONG) - input**
Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQBA_FIRST through MQBA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex **(MQLONG) - input**
This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength **(MQLONG) - input**
The length in bytes of the byte string contained in the *Buffer* parameter. The value must be zero or greater.

Buffer **(MQBYTE** × *BufferLength***) - input**
Buffer containing the byte string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetByteString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

C language invocation for mqSetByteString

```
mqSetByteString (Bag, Selector, ItemIndex, BufferLength, Buffer,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   BufferLength; /* Buffer length */
PMQBYTE  Buffer;       /* Buffer containing string */
MQLONG  CompCode;      /* Completion code */
MQLONG  Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetByteString

(Supported on Windows only.)

```
mqSetByteString Bag, Selector, ItemIndex, BufferLength, Buffer,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
```

```
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As Byte 'Buffer containing string'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqSetByteStringFilter:

The mqSetByteStringFilter call either modifies a byte string filter item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax for mqSetByteStringFilter

mqSetByteStringFilter *(Bag, Selector, ItemIndex, Bufferlength, Buffer, Operator, CompCode, Reason)*

Parameters for mqSetByteStringFilter

Bag **(MQHBAG) - input**
Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector **(MQLONG) - input**
Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQBA_FIRST through MQBA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex **(MQLONG) - input**
This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength **(MQLONG) - input**
The length in bytes of the condition byte string contained in the *Buffer* parameter. The value must be zero or greater.

Buffer **(MQBYTE** × *BufferLength***) - input**
Buffer containing the condition byte string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

Operator **(MQLONG** × *Operator***) - input**
Byte string filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetByteStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Bag handle not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

C language invocation for mqSetByteStringFilter

```
mqSetByteStringFilter (Bag, Selector, ItemIndex, BufferLength, Buffer,
Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   BufferLength; /* Buffer length */
PMQBYTE  Buffer;       /* Buffer containing string */
MQLONG   Operator;     /* Operator */
PMQLONG  CompCode;     /* Completion code */
PMQLONG  Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetByteStringFilter

(Supported on Windows only.)

```
mqSetByteStringFilter Bag, Selector, ItemIndex, BufferLength, Buffer,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As String 'Buffer containing string'
Dim Operator     As Long   'Item operator'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqSetInteger:

The mqSetInteger call either modifies an integer item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but specific system-data items can also be modified.

Syntax for mqSetInteger

mqSetInteger *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqSetInteger

Bag **(MQHBAG) - input**
Handle of the bag to be set. This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the handle you specify refers to a system bag.

Selector **(MQLONG) - input**
Selector of the item to be modified. If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read-only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex **(MQLONG) - input**
This value identifies the occurrence of the item with the specified selector that is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be one occurrence only of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

Note: For system selectors, the order is not changed.

ItemValue **(MQLONG) - input**
The integer value to be placed in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqSetInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not in valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read only and cannot be altered.

C language invocation for mqSetInteger

```
mqSetInteger (Bag, Selector, ItemIndex, ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Integer value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetInteger

(Supported on Windows only.)

```
mqSetInteger Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqSetInteger64:

The mqSetInteger64 call either modifies a 64-bit integer item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but specific system-data items can also be modified.

Syntax for mqSetInteger64

mqSetInteger64 *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters for mqSetInteger64

Bag **(MQHBAG) - input**
Handle of the bag to be set. This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the handle you specify refers to a system bag.

Selector **(MQLONG) - input**
Selector of the item to be modified. If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read-only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex **(MQLONG) - input**
This value identifies the occurrence of the item with the specified selector that is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be one occurrence only of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

Note: For system selectors, the order is not changed.

ItemValue **(MQINT64) - input**
The integer value to be placed in the bag.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqSetInteger64 call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not in valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read only and cannot be altered.

C language invocation for mqSetInteger64

```
mqSetInteger64 (Bag, Selector, ItemIndex, ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQINT64  ItemValue;     /* Integer value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetInteger64

(Supported on Windows only.)

```
mqSetInteger64 Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqSetIntegerFilter:

The mqSetIntegerFilter call either modifies an integer filter item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but specific system-data items can also be modified.

Syntax for mqSetIntegerFilter

mqSetIntegerFilter *(Bag, Selector, ItemIndex, ItemValue, Operator, CompCode, Reason)*

Parameters for mqSetIntegerFilter

Bag **(MQHBAG) - input**
Handle of the bag to be set. This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the handle you specify refers to a system bag.

Selector **(MQLONG) - input**
Selector of the item to be modified. If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read-only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the data type of the item must agree with the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex **(MQLONG) - input**
This value identifies the occurrence of the item with the specified selector that is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be one occurrence only of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

Note: For system selectors, the order is not changed.

ItemValue **(MQLONG) - input**
The integer condition value to be placed in the bag.

Operator **(MQLONG) - input**
The integer filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqSetIntegerFilter call:

MQRC_FILTER_OPERATOR_ERROR
Filter operator not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not in valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read only and cannot be altered.

C language invocation for mqSetIntegerFilter

```
mqSetIntegerFilter (Bag, Selector, ItemIndex, ItemValue, Operator,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Integer value */
```

```
MQLONG   Operator;      /* Item operator */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetIntegerFilter

(Supported on Windows only.)

```
mqSetIntegerFilter Bag, Selector, ItemIndex, ItemValue, Operator,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Integer value'
Dim Operator  As Long 'Item operator'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqSetString:

The mqSetString call either modifies a character data item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax for mqSetString

mqSetString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, CompCode, Reason)*

Parameters for mqSetString

Bag **(MQHBAG) - input**
Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector **(MQLONG) - input**
Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex **(MQLONG) - input**
This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength **(MQLONG) - input**
The length in bytes of the string contained in the *Buffer* parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED.

If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string.

If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present; the nulls do not delimit the string.

Buffer **(MQCHAR** × *BufferLength***) - input**
Buffer containing the character string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

Usage notes for mqSetString

The Coded Character Set ID (CCSID) associated with this string is copied from the current CCSID of the bag.

C language invocation for mqSetString

```
mqSetString (Bag, Selector, ItemIndex, BufferLength, Buffer,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   BufferLength; /* Buffer length */
PMQCHAR  Buffer;       /* Buffer containing string */
MQLONG  CompCode;      /* Completion code */
MQLONG  Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetString

(Supported on Windows only.)

```
mqSetString Bag, Selector, ItemIndex, BufferLength, Buffer,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As String 'Buffer containing string'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqSetStringFilter:

The mqSetStringFilter call either modifies a string filter item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax for mqSetStringFilter

mqSetStringFilter *(Bag, Selector, ItemIndex, Bufferlength, Buffer, Operator, CompCode, Reason)*

Parameters for mqSetStringFilter

Bag **(MQHBAG) - input**
Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector **(MQLONG) - input**
Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the data type of the item must be the same as the data type implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex **(MQLONG) - input**
This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described in this topic; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength **(MQLONG) - input**
The length in bytes of the condition string contained in the *Buffer* parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED.

If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string.

If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present; the nulls do not delimit the string.

Buffer **(MQCHAR** × *BufferLength***) - input**
Buffer containing the character condition string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

Operator **(MQLONG** × *Operator***) - input**
String filter operator to be placed in the bag. Valid operators are of the form MQCFOP_*.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetStringFilter call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_FILTER_OPERATOR_ERROR
Bag handle not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong data type for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

Usage notes for mqSetStringFilter

The Coded Character Set ID (CCSID) associated with this string is copied from the current CCSID of the bag.

C language invocation for mqSetStringFilter

```
mqSetStringFilter (Bag, Selector, ItemIndex, BufferLength, Buffer,
Operator, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   BufferLength; /* Buffer length */
PMQCHAR  Buffer;       /* Buffer containing string */
MQLONG  Operator;      /* Item operator */
MQLONG  CompCode;      /* Completion code */
MQLONG  Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqSetStringFilter

(Supported on Windows only.)

```
mqSetStringFilter Bag, Selector, ItemIndex, BufferLength, Buffer,
Operator, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As String 'Buffer containing string'
Dim Operator     As Long   'Item operator'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqTrim:

The mqTrim call trims the blanks from a blank-padded string, then terminates it with a null.

Syntax for mqTrim

mqTrim *(BufferLength, Buffer, String, CompCode, Reason)*

Parameters for mqTrim

BufferLength **(MQLONG) - input**
Length in bytes of the buffer containing the string padded with blanks. Must be zero or greater.

Buffer **(MQCHAR** × *BufferLength***) - input**
Buffer containing the blank-padded string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

String **(MQCHAR × (*BufferLength*+1)) - output**
Buffer to receive the null-terminated string. The length of this buffer must be at least one byte greater than the value of the *BufferLength* parameter.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqTrim call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_STRING_ERROR
String parameter not valid (invalid parameter address or buffer not completely accessible).

Usage notes for mqTrim

1. If the two buffer pointers are the same, the trimming is done in place. If they are not the same, the blank-padded string is copied into the null-terminated string buffer. After copying, the buffer is scanned backwards from the end until a nonspace character is found. The byte following the nonspace character is then overwritten with a null character.
2. If *String* and *Buffer* partially overlap, the result is undefined.

C language invocation for mqTrim

```
mqTrim (BufferLength, Buffer, String, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQLONG   BufferLength;     /* Buffer length */
PMQCHAR  Buffer;           /* Buffer containing blank-padded string */
MQCHAR   String[n+1];      /* String with blanks discarded */
MQLONG   CompCode;         /* Completion code */
MQLONG   Reason;           /* Reason code qualifying CompCode */
```

Note: This call is not supported in Visual Basic.

mqTruncateBag:

The mqTruncateBag call reduces the number of user items in a user bag to the specified value, by deleting user items from the end of the bag.

Syntax for mqTruncateBag

mqTruncateBag *(Bag, ItemCount, CompCode, Reason)*

Parameters for mqTruncateBag

Bag **(MQHBAG) - input**
Handle of the bag to be truncated. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

ItemCount **(MQLONG) - input**
The number of user items to remain in the bag after truncation. Zero is a valid value.

Note: The *ItemCount* parameter is the number of data items, not the number of unique selectors. (If there are one or more selectors that occur multiple times in the bag, there will be fewer selectors than data items before truncation.) Data items are deleted from the end of the bag, in the opposite order to which they were added to the bag.

If the number specified exceeds the number of user items currently in the bag, MQRC_ITEM_COUNT_ERROR results.

CompCode **(MQLONG) - output**
Completion code.

Reason **(MQLONG) - output**
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqTruncateBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_ITEM_COUNT_ERROR
ItemCount parameter not valid (value exceeds the number of user data items in the bag).

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes for mqTruncateBag

1. System items in a bag are not affected by mqTruncateBag; the call cannot be used to truncate system bags.
2. mqTruncateBag with an *ItemCount* of zero is not the same as the mqClearBag call. The former deletes all of the user items but leaves the system items intact, and the latter deletes all of the user items and resets the system items to their initial values.

C language invocation for mqTruncateBag

```
mqTruncateBag (Bag, ItemCount, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   ItemCount;     /* Number of items to remain in bag */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation for mqTruncateBag

(Supported on Windows only.)

```
mqTruncateBag Bag, ItemCount, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim ItemCount As Long 'Number of items to remain in bag'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

MQAI Selectors:

Items in bags are identified by a *selector* that acts as an identifier for the item. There are two types of selector, *user selector* and *system selector*.

User selectors

User selectors have values that are zero or positive. For the administration of MQSeries objects, valid user selectors are already defined by the following constants:

- MQCA_* and MQIA_* (object attributes)
- MQCACF_* and MQIACF_* (items relating specifically to PCF)
- MQCACH_* and MQIACH_* (channel attributes)

For user messages, the meaning of a user selector is defined by the application.

The following additional user selectors are introduced by the MQAI:

MQIACF_INQUIRY
Identifies a IBM WebSphere MQ object attribute to be returned by an Inquire command.

MQHA_BAG_HANDLE
Identifies a bag handle residing within another bag.

MQHA_FIRST
Lower limit for handle selectors.

MQHA_LAST
Upper limit for handle selectors.

MQHA_LAST_USED
Upper limit for last handle selector allocated.

MQCA_USER_LIST
Default user selector. Supported on Visual Basic only. This selector supports character type and represents the default value used if the *Selector* parameter is omitted on the mqAdd*, mqSet*, or mqInquire* calls.

MQIA_USER_LIST
Default user selector. Supported on Visual Basic only. This selector supports integer type and represents the default value used if the *Selector* parameter is omitted on the mqAdd*, mqSet*, or mqInquire* calls.

System selectors

System selectors have negative values. The following system selectors are included in the bag when it is created:

MQIASY_BAG_OPTIONS
Bag-creation options. A summation of the options used to create the bag. This selector cannot be changed by the user.

MQIASY_CODED_CHAR_SET_ID
Character-set identifier for the character data items in the bag. The initial value is the queue-manager's character set.

The value in the bag is used on entry to the mqExecute call and set on exit from the mqExecute call. This also applies when character strings are added to or modified in the bag.

MQIASY_COMMAND
PCF command identifier. Valid values are the MQCMD_* constants. For user messages, the value MQCMD_NONE should be used. The initial value is MQCMD_NONE.

The value in the bag is used on entry to the mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

MQIASY_COMP_CODE
Completion code. Valid values are the MQCC_* constants. The initial value is MQCC_OK.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_CONTROL
PCF control options. Valid values are the MQCFC_* constants. The initial value is MQCFC_LAST.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_MSG_SEQ_NUMBER
PCF message sequence number. Valid values are 1 or greater. The initial value is 1.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_REASON
Reason code. Valid values are the MQRC_* constants. The initial value is MQRC_NONE.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_TYPE
PCF command type. Valid values are the MQCFT_* constants. For user messages, the value MQCFT_USER should be used. The initial value is MQCFT_USER for bags created as user bags and MQCFT_COMMAND for bags created as administration or command bags.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_VERSION
PCF version. Valid values are the MQCFH_VERSION_* constants. The initial value is MQCFH_VERSION_1.

If the value in the bag is set to a value other than MQCFH_VERSION_1, the value is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls. If the value in the bag in MQCFH_VERSION_1, the PCF version is the lowest value required for the parameter structures that are present in the message.

The value in the bag is set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

Indexing

Each selector and value within a data item in a bag have three associated index numbers:

- The index relative to other items that have the same selector.
- The index relative to the category of selector (user or system) to which the item belongs.
- The index relative to all the data items in the bag (user and system).

This allows indexing by user selectors, system selectors, or both as shown in Figure 7 on page 1333.

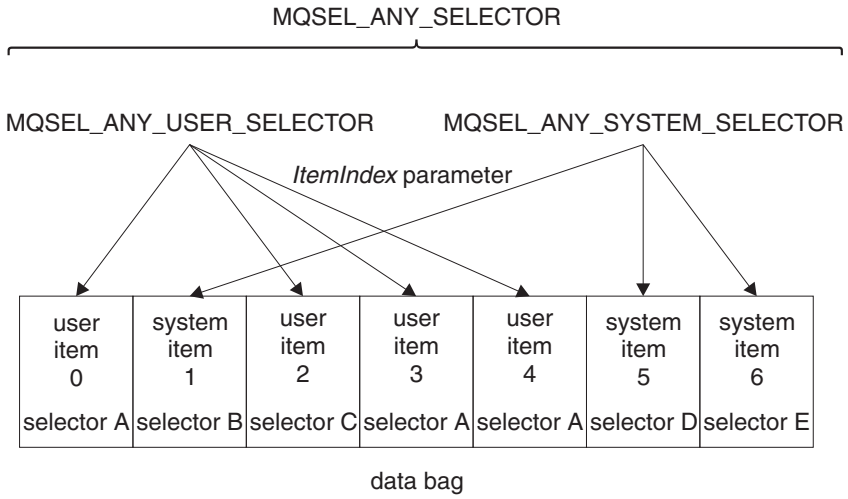


Figure 7. Indexing

In Figure Figure 7, user item 3 (selector A) can be referred to by the following index pairs:

Selector	*ItemIndex*
selector A	1
MQSEL_ANY_USER_SELECTOR	2
MQSEL_ANY_SELECTOR	3

The index is zero-based like an array in C; if there are 'n' occurrences, the index ranges from zero through 'n-1', with no gaps.

Indexes are used when replacing or removing existing data items from a bag. When used in this way, the insertion order is preserved, but indexes of other data items can be affected. For examples of this, see Changing information within a bag and Deleting data items.

The three types of indexing allow easy retrieval of data items. For example, if there are three instances of a particular selector in a bag, the mqCountItems call can count the number of instances of that selector, and the mqInquire* calls can specify both the selector and the index to inquire those values only. This is useful for attributes that can have a list of values such as some of the exits on channels.

Data conversion

Like PCF messages, the strings contained in an MQAI data bag can be in a variety of coded character sets. Usually, all of the strings in a PCF message are in the same coded character set; that is, the same set as the queue manager.

Each string item in a data bag contains two values; the string itself and the CCSID. The string that is added to the bag is obtained from the *Buffer* parameter of the mqAddString or mqSetString call. The CCSID is obtained from the system item containing a selector of MQIASY_CODED_CHAR_SET_ID. This is known as the *bag CCSID* and can be changed using the mqSetInteger call.

When you inquire the value of a string contained in a data bag, the CCSID is an output parameter from the call.

Table 106 on page 1334 shows the rules applied when converting data bags into messages and vice versa:

Table 106. CCSID processing

MQAI call	CCSID	Input to call	Output to call
mqBagToBuffer	Bag CCSID (1)	Ignored	Unchanged
mqBagToBuffer	String CCSIDs in bag	Used	Unchanged
mqBagToBuffer	String CCSIDs in buffer	Not applicable	Copied from string CCSIDs in bag
mqBufferToBag	Bag CCSID (1)	Ignored	Unchanged
mqBufferToBag	String CCSIDs in buffer	Used	Unchanged
mqBufferToBag	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in buffer
mqPutBag	MQMD CCSID	Used	Unchanged (2)
mqPutBag	Bag CCSID (1)	Ignored	Unchanged
mqPutBag	String CCSIDs in bag	Used	Unchanged
mqPutBag	String CCSIDs in message sent	Not applicable	Copied from string CCSIDs in bag
mqGetBag	MQMD CCSID	Used for data conversion of message	Set to CCSID of data returned (3)
mqGetBag	Bag CCSID (1)	Ignored	Unchanged
mqGetBag	String CCSIDs in message	Used	Unchanged
mqGetBag	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in message
mqExecute	Request-bag CCSID	Used for MQMD of request message (4)	Unchanged
mqExecute	Reply-bag CCSID	Used for data conversion of reply message (4)	Set to CCSID of data returned (3)
mqExecute	String CCSIDs in request bag	Used for request message	Unchanged
mqExecute	String CCSIDs in reply bag	Not applicable	Copied from string CCSIDs in reply message

Notes:

1. Bag CCSID is the system item with selector MQIASY_CODED_CHAR_SET_ID.
2. MQCCSI_Q_MGR is changed to the actual queue manager CCSID.
3. If data conversion is requested, the CCSID of data returned is the same as the output value. If data conversion is not requested, the CCSID of data returned is the same as the message value. Note that no message is returned if data conversion is requested but fails.
4. If the CCSID is MQCCSI_DEFAULT, the queue manager's CCSID is used.

Use of the message descriptor

The PCF command type is obtained from the system item with selector MQIASY_TYPE. When you create your data bag, the initial value of this item is set depending on the type of bag you create:

Table 107. PCF command type

Type of bag	Initial value of MQIASY_TYPE item
MQCBO_ADMIN_BAG	MQCFT_COMMAND
MQCBO_COMMAND_BAG	MQCFT_COMMAND
MQCBO_*	MQCFT_USER

When the MQAI generates a message descriptor, the values used in the *Format* and *MsgType* parameters depend on the value of the system item with selector MQIASY_TYPE as shown in Table 107.

Table 108. Format and MsgType parameters of the MQMD

PCF command type	Format	MsgType
MQCFT_COMMAND	MQFMT_ADMIN	MQMT_REQUEST
MQCFT_REPORT	MQFMT_ADMIN	MQMT_REPORT
MQCFT_RESPONSE	MQFMT_ADMIN	MQMT_REPLY
MQCFT_TRACE_ROUTE	MQFMT_ADMIN	MQMT_DATAGRAM
MQCFT_EVENT	MQFMT_EVENT	MQMT_DATAGRAM
MQCFT_*	MQFMT_PCF	MQMT_DATAGRAM

Table 108 shows that if you create an administration bag or a command bag, the *Format* of the message descriptor is MQFMT_ADMIN and the *MsgType* is MQMT_REQUEST. This is suitable for a PCF request message sent to the command server when a response is expected back.

Other parameters in the message descriptor take the values shown in Table 109.

Table 109. Message descriptor values

Parameter	Value
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	see Table 108
Expiry	30 seconds (note 1 on page 1336)
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
CodedCharSetId	depends on the bag CCSID (note 2 on page 1336)
Format	see Table 108
Priority	MQPRI_PRIORITY_AS_Q_DEF
Persistence	MQPER_NOT_PERSISTENT
MsgId	MQMI_NONE
CorrelId	MQCI_NONE
BackoutCount	0
ReplyToQ	see note 3 on page 1336
ReplyToQMgr	blank

Table 109. Message descriptor values (continued)

Parameter	Value
Notes: 1. This value can be overridden on the mqExecute call by using the *OptionsBag* parameter. For information about this, see "mqExecute" on page 1282. 2. See "Data conversion" on page 1333. 3. Name of the user-specified reply queue or MQAI-generated temporary dynamic queue for messages of type MQMT_REQUEST. Blank otherwise.	

Example code

Here are some example uses of the mqExecute call.

The example shown in figure Figure 8 creates a local queue (with a maximum message length of 100 bytes) on a queue manager:

```
/* Create a bag for the data you want in your PCF message */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagRequest)

/* Create a bag to be filled with the response from the command server */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagResponse)

/* Create a queue    */
/* Supply queue name */
mqAddString(hbagRequest, MQCA_Q_NAME, "QBERT")

/* Supply queue type */
mqAddString(hbagRequest, MQIA_Q_TYPE, MQQT_LOCAL)

/* Maximum message length is an optional parameter */
mqAddString(hbagRequest, MQIA_MAX_MSG_LENGTH, 100)

/* Ask the command server to create the queue */
mqExecute(MQCMD_CREATE_Q, hbagRequest, hbagResponse)

/* Tidy up memory allocated */
mqDeleteBag(hbagRequest)
mqDeleteBag(hbagResponse)
```

Figure 8. Using mqExecute to create a local queue

The example shown in figure Figure 9 on page 1337 inquires about all attributes of a particular queue. The mqAddInquiry call identifies all WebSphere MQ object attributes of a queue to be returned by the Inquire parameter on mqExecute.

```
/* Create a bag for the data you want in your PCF message */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagRequest)

/* Create a bag to be filled with the response from the command server */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagResponse)

/* Inquire about a queue by supplying its name */
/* (other parameters are optional) */
mqAddString(hbagRequest, MQCA_Q_NAME, "QBERT")

/* Request the command server to inquire about the queue */
mqExecute(MQCMD_INQUIRE_Q, hbagRequest, hbagResponse)

/* If it worked, the attributes of the queue are returned */
/* in a system bag within the response bag */
mqInquireBag(hbagResponse, MQHA_BAG_HANDLE, 0, &hbagAttributes)

/* Inquire the name of the queue and its current depth */
mqInquireString(hbagAttributes, MQCA_Q_NAME, &stringAttribute)
mqInquireString(hbagAttributes, MQIA_CURRENT_Q_DEPTH, &integerAttribute)

/* Tidy up memory allocated */
mqDeleteBag(hbagRequest)
mqDeleteBag(hbagResponse)
```

Figure 9. Using mqExecute to inquire about queue attributes

Using mqExecute is the simplest way of administering WebSphere MQ, but lower-level calls, mqBagToBuffer and mqBufferToBag, can be used. For more information about the use of these calls, see Introduction to the WebSphere MQ Administration Interface (MQAI).

For sample programs, see Examples of using the MQAI.

Developing applications reference

Use the links provided in this section to help you develop your WebSphere MQ applications:

- "MQI applications reference"
- "User exits, API exits, and installable services reference" on page 2325
- "SOAP reference" on page 2273
- "Reference material for WebSphere MQ bridge for HTTP" on page 2542
- "The WebSphere MQ .NET classes and interfaces" on page 2578
- "WebSphere MQ C++ classes" on page 2639
- "The WebSphere MQ classes for Java libraries" on page 2748
- WebSphere MQ classes for JMS

Related information:

Developing applications

MQI applications reference

Use the links provided in this section to help you develop your MQI applications:

- "Code examples" on page 1338
- "Constants" on page 1388
- "Data types used in the MQI" on page 1529
- "Function calls" on page 1925
- "Attributes of objects" on page 2096
- "Return codes" on page 2178
- "Rules for validating MQI options" on page 2179

- "Machine encodings" on page 2203
- "Report options and message flags" on page 2206
- "Data conversion" on page 2210
- "Properties specified as MQRFH2 elements" on page 2233
- "Code page conversion" on page 2241

Related concepts:

"User exits, API exits, and installable services reference" on page 2325
Use the links provided in this section to help you develop your User exits, API exits, and installable services applications:

"The WebSphere MQ classes for Java libraries" on page 2748
The location of the WebSphere MQ classes for Java libraries varies according to platform. Specify this location when you start an application.

Related reference:

"SOAP reference" on page 2273
WebSphere MQ transport for SOAP reference information arranged alphabetically.

"Reference material for WebSphere MQ bridge for HTTP" on page 2542
Reference topics for WebSphere MQ bridge for HTTP, arranged alphabetically

"The WebSphere MQ .NET classes and interfaces" on page 2578
WebSphere MQ .NET classes and interfaces are listed alphabetically. The properties, methods and constructors are described.

"WebSphere MQ C++ classes" on page 2639
The WebSphere MQ C++ classes encapsulate the WebSphere MQ Message Queue Interface (MQI). There is a single C++ header file, **imqi.hpp**, which covers all of these classes.

Related information:

Developing applications

../com.ibm.mq.javadoc.doc/WMQJMSClasses/index.html

Code examples

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

C language examples:

This collection of topics is mostly taken from the WebSphere MQ for z/OS sample applications. They are applicable to all platforms, except where noted.

Connecting to a queue manager:

This example demonstrates how to use the MQCONN call to connect a program to a queue manager in z/OS batch.

This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
#include <cmqc.h>
.
.
.
static char Parm1[MQ_Q_MGR_NAME_LENGTH] ;
.
.
.
int  main(int argc, char *argv[] )
   {
   /*                                                  */
   /*     Variables for MQ calls                       */
   /*                                                  */
   MQHCONN Hconn;      /* Connection handle            */
   MQLONG  CompCode;   /* Completion code              */
```

```
   MQLONG  Reason;     /* Qualifying reason             */
.
.
.
   /* Copy the queue manager name, passed in the        */
   /* parm field, to Parm1                              */
   strncpy(Parm1,argv[1],MQ_Q_MGR_NAME_LENGTH);
.
.
.
   /*                                                   */
   /* Connect to the specified queue manager.           */
   /*   Test the output of the connect call.  If the    */
   /*   call fails, print an error message showing the  */
   /*   completion code and reason code, then leave the */
   /*   program.                                        */
   /*                                                   */
   MQCONN(Parm1,
          &Hconn,
          &CompCode,
          &Reason);
   if ((CompCode != MQCC_OK) | (Reason != MQRC_NONE))
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQCONN, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      goto AbnormalExit2;
      }
.
.
.
   }
```

Disconnecting from a queue manager:

This example demonstrates how to use the MQDISC call to disconnect a program from a queue manager in z/OS batch.

The variables used in this code extract are those that were set in "Connecting to a queue manager" on page 1338. This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
.
.
.
   /*                                                   */
   /*   Disconnect from the queue manager.  Test the    */
   /*   output of the disconnect call.  If the call     */
   /*   fails, print an error message showing the       */
   /*   completion code and reason code.                */
   /*                                                   */
   MQDISC(&Hconn,
          &CompCode,
          &Reason);
   if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQDISC, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      }
.
.
.
```

Creating a dynamic queue:

This example demonstrates how to use the MQOPEN call to create a dynamic queue.

This extract is taken from the Mail Manager sample application (program CSQ4TCD1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
.
.
.
MQLONG  HCONN = 0;   /* Connection handle      */
MQHOBJ  HOBJ;        /* MailQ Object handle    */
MQHOBJ  HobjTempQ;   /* TempQ Object Handle    */
MQLONG  CompCode;    /* Completion code        */
MQLONG  Reason;      /* Qualifying reason      */
MQOD    ObjDesc = {MQOD_DEFAULT};
                     /* Object descriptor      */
MQLONG  OpenOptions; /* Options control MQOPEN */
.
.
.
   /*---------------------------------------- */
   /* Initialize the Object Descriptor (MQOD)  */
   /* control block.  (The remaining fields    */
   /* are already initialized.)                */
   /*----------------------------------------*/
   strncpy( ObjDesc.ObjectName,
            SYSTEM_REPLY_MODEL,
            MQ_Q_NAME_LENGTH );
   strncpy( ObjDesc.DynamicQName,
            SYSTEM_REPLY_INITIAL,
            MQ_Q_NAME_LENGTH );
   OpenOptions = MQOO_INPUT_AS_Q_DEF;
   /*----------------------------------------*/
   /* Open the model queue and, therefore,     */
   /* create and open a temporary dynamic      */
   /* queue                                    */
   /*----------------------------------------*/
   MQOPEN( HCONN,
           &ObjDesc,
           OpenOptions,
           &HobjTempQ,
           &CompCode,
           &Reason );
   if ( CompCode == MQCC_OK ) {
.
.
.
   }
   else {
      /*---------------------------------------*/
      /* Build an error message to report the  */
      /* failure of the opening of the model   */
      /* queue                                 */
      /*---------------------------------------*/
      MQMErrorHandling( "OPEN TEMPQ", CompCode,
                        Reason );
      ErrorFound = TRUE;
   }
   return ErrorFound;
}
.
.
.
```

Opening an existing queue:

This example demonstrates how to use the MQOPEN call to open a queue that has already been defined.

This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
#include <cmqc.h>
.
.
.
static char Parm1[MQ_Q_MGR_NAME_LENGTH];
.
.
.
int  main(int argc, char *argv[] )
   {
   /*
   /*     Variables for MQ calls                          */
   /*
   MQHCONN Hconn ;            /* Connection handle       */
   MQLONG  CompCode;          /* Completion code         */
   MQLONG  Reason;            /* Qualifying reason       */
   MQOD    ObjDesc = { MQOD_DEFAULT };
                              /* Object descriptor       */
   MQLONG  OpenOptions;       /* Options that control    */
                              /*  the MQOPEN call        */
   MQHOBJ  Hobj;              /* Object handle           */
.
.
.
   /*  Copy the queue name, passed in the parm field,    */
   /*  to Parm2 strncpy(Parm2,argv[2],         */
   /*  MQ_Q_NAME_LENGTH);                                */
.
.
.
   /*                                                    */
   /*  Initialize the object descriptor (MQOD) control   */
   /*  block.  (The initialization default sets StrucId, */
   /*  Version, ObjectType, ObjectQMgrName,              */
   /*  DynamicQName, and AlternateUserid fields)         */
   /*                                                    */
   strncpy(ObjDesc.ObjectName,Parm2,MQ_Q_NAME_LENGTH);
.
.
.
   /*  Initialize the other fields required for the open */
   /*  call (Hobj is set by the MQCONN call).            */
   /*                                                    */
   OpenOptions  = MQOO_BROWSE;
.
.
.
   /*                                                    */
   /* Open the queue.                                    */
   /*   Test the output of the open call.  If the call   */
   /*   fails, print an error message showing the        */
   /*   completion code and reason code, then bypass     */
   /*   processing, disconnect and leave the program.    */
   /*                                                    */
   MQOPEN(Hconn,
          &ObjDesc,
          OpenOptions,
          &Hobj,
          &CompCode,
          &Reason);
   if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQOPEN, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      goto AbnormalExit1;       /* disconnect processing */
      }
```

```
   } /* end of main */
```

Closing a queue:

This example demonstrates how to use the MQCLOSE call to close a queue.

This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
   /*                                                  */
   /*   Close the queue.                               */
   /*   Test the output of the close call.  If the call  */
   /*   fails, print an error message showing the        */
   /*   completion code and reason code.               */
   /*                                                  */
   MQCLOSE(Hconn,
           &Hobj,
           MQCO_NONE,
           &CompCode,
           &Reason);
   if ((CompCode != MQCC_OK) || (Reason != MQRC_NONE))
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQCLOSE, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      }
```

Putting a message using MQPUT:

This example demonstrates how to use the MQPUT call to put a message on a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ. For the names and locations of the sample applications, see Sample programs (platforms except z/OS).

```
qput()
{
     MQMD    MsgDesc;
     MQPMO   PutMsgOpts;
     MQLONG  CompCode;
     MQLONG  Reason;
     MQHCONN Hconn;
     MQHOBJ  Hobj;
     char message_buffer[] = "MY MESSAGE";
    /*-------------------------------*/
    /*  Set up PMO structure.        */
    /*-------------------------------*/
     memset(&PutMsgOpts, '\0', sizeof(PutMsgOpts));
     memcpy(PutMsgOpts.StrucId, MQPMO_STRUC_ID,
            sizeof(PutMsgOpts.StrucId));
     PutMsgOpts.Version = MQPMO_VERSION_1;
     PutMsgOpts.Options = MQPMO_SYNCPOINT;

    /*-------------------------------*/
    /*  Set up MD structure.         */
    /*-------------------------------*/
    memset(&MsgDesc, '\0', sizeof(MsgDesc));
    memcpy(MsgDesc.StrucId, MQMD_STRUC_ID,
```

```
       sizeof(MsgDesc.StrucId));
    MsgDesc.Version     =  MQMD_VERSION_1;
    MsgDesc.Expiry      =  MQEI_UNLIMITED;
    MsgDesc.Report      =  MQRO_NONE;
    MsgDesc.MsgType     =  MQMT_DATAGRAM;
    MsgDesc.Priority    = 1;
    MsgDesc.Persistence = MQPER_PERSISTENT;
    memset(MsgDesc.ReplyToQ,
           '\0',
           sizeof(MsgDesc.ReplyToQ));
    /*-----------------------------------------------------*/
    /*  Put the message.                                   */
    /*-----------------------------------------------------*/
    MQPUT(Hconn, Hobj, &MsgDesc, &PutMsgOpts,
          sizeof(message_buffer), message_buffer,
          &CompCode, &Reason);

    /*-------------------------------------*/
    /*  Check completion and reason codes. */
    /*-------------------------------------*/
    switch (CompCode)
        {
         case MQCC_OK:
             break;
         case MQCC_FAILED:
             switch (Reason)
                 {
                  case MQRC_Q_FULL:
                  case MQRC_MSG_TOO_BIG_FOR_Q:
                      break;
                  default:
                      break; /* Perform error processing */
                  }
             break;
         default:
             break;          /* Perform error processing */
        }
}
```

Putting a message using MQPUT1:

This example demonstrates how to use the MQPUT1 call to open a queue, put a single message on the queue, then close the queue.

This extract is taken from the Credit Check sample application (program CSQ4CCB5) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
⋮
MQLONG   Hconn;              /* Connection handle        */
MQHOBJ   Hobj_CheckQ;        /* Object handle            */
MQLONG   CompCode;           /* Completion code          */
MQLONG   Reason;             /* Qualifying reason        */
MQOD     ObjDesc    = {MQOD_DEFAULT};
                             /* Object descriptor        */
MQMD     MsgDesc    = {MQMD_DEFAULT};
                             /* Message descriptor       */
MQLONG   OpenOptions;        /* Control the MQOPEN call  */

MQGMO    GetMsgOpts = {MQGMO_DEFAULT};
                             /* Get Message Options      */
MQLONG   MsgBuffLen;         /* Length of message buffer */
CSQ4BCAQ MsgBuffer;          /* Message structure        */
MQLONG   DataLen;            /* Length of message        */
```

```
MQPMO    PutMsgOpts = {MQPMO_DEFAULT};
                            /* Put Message Options      */
CSQ4BQRM PutBuffer;         /* Message structure        */
MQLONG   PutBuffLen = sizeof(PutBuffer);
                            /* Length of message buffer */
.
.
.
void Process_Query(void)
   {
   /*                                                    */
   /* Build the reply message                            */
   /*                                                    */
.
.
.
   /*                                                    */
   /* Set the object descriptor, message descriptor and  */
   /* put message options to the values required to      */
   /* create the reply message.                          */
   /*                                                    */
   strncpy(ObjDesc.ObjectName, MsgDesc.ReplyToQ,
           MQ_Q_NAME_LENGTH);
   strncpy(ObjDesc.ObjectQMgrName, MsgDesc.ReplyToQMgr,
           MQ_Q_MGR_NAME_LENGTH);
   MsgDesc.MsgType = MQMT_REPLY;
   MsgDesc.Report  = MQRO_NONE;
   memset(MsgDesc.ReplyToQ, ' ', MQ_Q_NAME_LENGTH);
   memset(MsgDesc.ReplyToQMgr, ' ', MQ_Q_MGR_NAME_LENGTH);
   memcpy(MsgDesc.MsgId, MQMI_NONE, sizeof(MsgDesc.MsgId));
   PutMsgOpts.Options = MQPMO_SYNCPOINT +
                        MQPMO_PASS_IDENTITY_CONTEXT;
   PutMsgOpts.Context = Hobj_CheckQ;
   PutBuffLen = sizeof(PutBuffer);
   MQPUT1(Hconn,
          &ObjDesc,
          &MsgDesc,
          &PutMsgOpts,
          PutBuffLen,
          &PutBuffer,
          &CompCode,
          &Reason);

   if (CompCode != MQCC_OK)
      {
      strncpy(TS_Operation, "MQPUT1",
              sizeof(TS_Operation));
      strncpy(TS_ObjName, ObjDesc.ObjectName,
              MQ_Q_NAME_LENGTH);
      Record_Call_Error();
      Forward_Msg_To_DLQ();
      }
   return;
   }
.
.
.
```

Getting a message:

This example demonstrates how to use the MQGET call to remove a message from a queue.

This extract is taken from the Browse sample application (program CSQ4BCA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
#include "cmqc.h"
.
.
.
#define BUFFERLENGTH 80
.
.
.
int  main(int argc, char *argv[] )
```

```
   {
   /*                                                  */
   /*     Variables for MQ calls                       */
   /*                                                  */
   MQHCONN Hconn ;            /* Connection handle      */
   MQLONG  CompCode;          /* Completion code        */
   MQLONG  Reason;            /* Qualifying reason      */
   MQHOBJ  Hobj;              /* Object handle          */
   MQMD    MsgDesc  = { MQMD_DEFAULT };
                              /* Message descriptor     */
   MQLONG  DataLength ;       /* Length of the message  */
   MQCHAR  Buffer[BUFFERLENGTH+1];
                              /* Area for message data  */
   MQGMO   GetMsgOpts = { MQGMO_DEFAULT };
                              /* Options which control  */
                              /*  the MQGET call        */
   MQLONG  BufferLength = BUFFERLENGTH ;
                              /* Length of buffer       */
.
.
.
   /*     No need to change the message descriptor     */
   /*     (MQMD) control block because initialization  */
   /*     default sets all the fields.                 */
   /*                                                  */
   /*     Initialize the get message options (MQGMO)   */
   /*     control block (the copy file initializes all */
   /*     the other fields).                           */
   /*                                                  */
   GetMsgOpts.Options = MQGMO_NO_WAIT      +
                        MQGMO_BROWSE_FIRST +
                        MQGMO_ACCEPT_TRUNCATED_MSG;
   /*                                                  */
   /* Get the first message.                           */
   /*   Test for the output of the call is carried out */
   /*   in the 'for' loop.                             */
   /*                                                  */
   MQGET(Hconn,
         Hobj,
         &MsgDesc,
         &GetMsgOpts,
         BufferLength,
         Buffer,
         &DataLength,
         &CompCode,
         &Reason);
   /*                                                  */
   /* Process the message and get the next message,    */
   /* until no messages remaining.                     */
.
.
.
   /*     If the call fails for any other reason,      */
   /*     print an error message showing the completion*/
   /*     code and reason code.                        */
   /*                                                  */
   if ( (CompCode == MQCC_FAILED) &&
       (Reason   == MQRC_NO_MSG_AVAILABLE) )
      {
.
.
.
      }
   else
      {
      sprintf(pBuff, MESSAGE_4_E,
              ERROR_IN_MQGET, CompCode, Reason);
      PrintLine(pBuff);
      RetCode = CSQ4_ERROR;
      }
.
.
.
   } /* end of main */
```

Getting a message using the wait option:

This example demonstrates how to use the wait option of the MQGET call.

This code accepts truncated messages. This extract is taken from the Credit Check sample application (program CSQ4CCB5) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
.
.
.
MQLONG   Hconn;              /* Connection handle        */
MQHOBJ   Hobj_CheckQ;        /* Object handle            */
MQLONG   CompCode;           /* Completion code          */
MQLONG   Reason;             /* Qualifying reason        */
MQOD     ObjDesc    = {MQOD_DEFAULT};
                             /* Object descriptor        */
MQMD     MsgDesc    = {MQMD_DEFAULT};
                             /* Message descriptor       */
MQLONG   OpenOptions;
                             /* Control the MQOPEN call  */
MQGMO    GetMsgOpts = {MQGMO_DEFAULT};
                             /* Get Message Options      */
MQLONG   MsgBuffLen;         /* Length of message buffer */
CSQ4BCAQ MsgBuffer;          /* Message structure        */
MQLONG   DataLen;            /* Length of message        */
.
.
.
void main(void)
   {
.
.
.
   /*                                                    */
   /* Initialize options and open the queue for input    */
   /*                                                    */
.
.
.
      /*                                                 */
      /* Get and process messages                        */
      /*                                                 */
      GetMsgOpts.Options = MQGMO_WAIT +
                           MQGMO_ACCEPT_TRUNCATED_MSG +
                           MQGMO_SYNCPOINT;
      GetMsgOpts.WaitInterval = WAIT_INTERVAL;
      MsgBuffLen = sizeof(MsgBuffer);
      memcpy(MsgDesc.MsgId, MQMI_NONE,
             sizeof(MsgDesc.MsgId));
      memcpy(MsgDesc.CorrelId, MQCI_NONE,
             sizeof(MsgDesc.CorrelId));
      /*                                                 */
      /* Make the first MQGET call outside the loop      */
      /*                                                 */
      MQGET(Hconn,
            Hobj_CheckQ,
            &MsgDesc,
            &GetMsgOpts,
            MsgBuffLen,
            &MsgBuffer,
            &DataLen,
            &CompCode,
            &Reason);
.
.
.
      /*                                                 */
      /* Test the output of the MQGET call.  If the call */
      /* failed, send an error message showing the       */
      /* completion code and reason code, unless the     */
      /* reason code is NO_MSG AVAILABLE.                */
      /*                                                 */
      if (Reason != MQRC_NO_MSG_AVAILABLE)
```

```
        {
        strncpy(TS_Operation, "MQGET", sizeof(TS_Operation));
        strncpy(TS_ObjName, ObjDesc.ObjectName,
                MQ_Q_NAME_LENGTH);
        Record_Call_Error();
        }
⋮
```

Getting a message using signaling:

Signaling is available only with WebSphere MQ for z/OS.

This example demonstrates how to use the MQGET call to set a signal so that you are notified when a suitable message arrives on a queue. This extract is not taken from the sample applications supplied with WebSphere MQ.

```
⋮
get_set_signal()
{
     MQMD    MsgDesc;
     MQGMO   GetMsgOpts;
     MQLONG  CompCode;
     MQLONG  Reason;
     MQHCONN Hconn;
     MQHOBJ  Hobj;
     MQLONG  BufferLength;
     MQLONG  DataLength;
     char message_buffer[100];
     long  int q_ecb, work_ecb;
     short int signal_sw, endloop;
     long  int mask = 255;

     /*---------------------------*/
     /*  Set up GMO structure.    */
     /*---------------------------*/
     memset(&GetMsgOpts,'\0',sizeof(GetMsgOpts));
     memcpy(GetMsgOpts.StrucId, MQGMO_STRUC_ID,
            sizeof(GetMsgOpts.StrucId);
     GetMsgOpts.Version      = MQGMO_VERSION_1;
     GetMsgOpts.WaitInterval = 1000;
     GetMsgOpts.Options      = MQGMO_SET_SIGNAL +
                               MQGMO_BROWSE_FIRST;
     q_ecb              = 0;
     GetMsgOpts.Signal1      = &q_ecb;
     /*---------------------------*/
     /*  Set up MD structure.     */
     /*---------------------------*/
     memset(&MsgDesc,'\0',sizeof(MsgDesc));
     memcpy(MsgDesc.StrucId, MQMD_STRUC_ID,
            sizeof(MsgDesc.StrucId);
     MsgDesc.Version  = MQMD_VERSION_1;
     MsgDesc.Report   = MQRO_NONE;
     memcpy(MsgDesc.MsgId,MQMI_NONE,
            sizeof(MsgDesc.MsgId));
     memcpy(MsgDesc.CorrelId,MQCI_NONE,
            sizeof(MsgDesc.CorrelId));
    /*-------------------------------------------------*/
    /*  Issue the MQGET call.                          */
    /*-------------------------------------------------*/
     BufferLength = sizeof(message_buffer);
     signal_sw   = 0;

     MQGET(Hconn, Hobj, &MsgDesc, &GetMsgOpts,
           BufferLength, message_buffer, &DataLength,
           &CompCode, &Reason);
```

```
/*-----------------------------------*/
/*  Check completion and reason codes. */
/*-----------------------------------*/
 switch (CompCode)
     {
      case (MQCC_OK):          /* Message retrieved  */
          break;
      case (MQCC_WARNING):
          switch (Reason)
          {
           case (MQRC_SIGNAL_REQUEST_ACCEPTED):
               signal_sw = 1;
               break;
           default:
               break;   /* Perform error processing  */
          }
          break;
      case (MQCC_FAILED):
          switch (Reason)
          {
           case (MQRC_Q_MGR_NOT_AVAILABLE):
           case (MQRC_CONNECTION_BROKEN):
           case (MQRC_Q_MGR_STOPPING):
               break;
           default:
               break;  /* Perform error processing.  */
          }
          break;
      default:
          break;       /* Perform error processing.  */
     }
/*-------------------------------------------------*/
/* If the SET_SIGNAL was accepted, set up a loop to  */
/* check whether a message has arrived at one second */
/* intervals. The loop ends if a message arrives or  */
/* the wait interval specified in the MQGMO          */
/* structure has expired.                            */
/*                                                   */
/* If a message arrives on the queue, another MQGET  */
/* must be issued to retrieve the message. If other  */
/* MQM calls have been made in the intervening       */
/* period, this may necessitate reinitializing the   */
/* MQMD and MQGMO structures.                        */
/* In this code, no intervening calls                */
/* have been made, so the only change required to    */
/* the structures is to specify MQGMO_NO_WAIT,       */
/* since we now know the message is there.           */
/*                                                   */
/* This code uses the EXEC CICS DELAY command to     */
/* suspend the program for a second. A batch program */
/* may achieve the same effect by calling an         */
/* assembler language subroutine which issues a      */
/* z/OS STIMER macro.                                */
/*-------------------------------------------------*/
if (signal_sw == 1)
   {
    endloop = 0;
    do
      {
       EXEC CICS DELAY FOR HOURS(0) MINUTES(0) SECONDS(1);
       work_ecb = q_ecb & mask;
       switch (work_ecb)
           {
            case (MQEC_MSG_ARRIVED):
                endloop = 1;
                mqgmo_options = MQGMO_NO_WAIT;
                MQGET(Hconn, Hobj, &MsgDesc, &GetMsgOpts,
```

```
                        BufferLength, message_buffer,
                        &DataLength, &CompCode, &Reason);
                    if (CompCode != MQCC_OK)
                        ;          /* Perform error processing. */
                    break;
                case (MQEC_WAIT_INTERVAL_EXPIRED):
                case (MQEC_WAIT_CANCELED):
                    endloop = 1;
                    break;
                default:
                    break;
               }
          } while (endloop == 0);
       }
     return;
}
```

Inquiring about the attributes of an object:

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CCC1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
#include <cmqc.h>      /* MQ API header file       */
:
#define NUMBEROFSELECTORS  2

const MQHCONN Hconn = MQHC_DEF_HCONN;
:
static void InquireGetAndPut(char   *Message,
                             PMQHOBJ pHobj,
                             char   *Object)

   {
   /*      Declare local variables                       */
   /*                                                    */
   MQLONG  SelectorCount = NUMBEROFSELECTORS;
                                 /* Number of selectors  */
   MQLONG  IntAttrCount  = NUMBEROFSELECTORS;
                                 /* Number of int attrs  */
   MQLONG  CharAttrLength = 0;
                    /* Length of char attribute buffer  */
   MQCHAR *CharAttrs ;
                    /* Character attribute buffer       */
   MQLONG  SelectorsTable[NUMBEROFSELECTORS];
                                 /* attribute selectors  */
   MQLONG  IntAttrsTable[NUMBEROFSELECTORS];
                                 /* integer attributes   */
   MQLONG  CompCode;             /* Completion code      */
   MQLONG  Reason;               /* Qualifying reason    */
   /*                                                    */
   /*     Open the queue.  If successful, do the inquire */
   /*     call.                                          */
   /*                                                    */
      /*                                                 */
      /*   Initialize the variables for the inquire      */
      /*   call:                                         */
      /*    - Set SelectorsTable to the attributes whose */
      /*      status is                                  */
      /*       required                                  */
      /*    - All other variables are already set        */
      /*                                                 */
      SelectorsTable[0] = MQIA_INHIBIT_GET;
      SelectorsTable[1] = MQIA_INHIBIT_PUT;
```

```
      /*                                                  */
      /*   Issue the inquire call                         */
      /*     Test the output of the inquire call. If the */
      /*     call failed, display an error message       */
      /*     showing the completion code and reason code,*/
      /*     otherwise display the status of the         */
      /*     INHIBIT-GET and INHIBIT-PUT attributes      */
      /*                                                  */
      MQINQ(Hconn,
            *pHobj,
            SelectorCount,
            SelectorsTable,
            IntAttrCount,
            IntAttrsTable,
            CharAttrLength,
            CharAttrs,
            &CompCode,
            &Reason);
      if (CompCode != MQCC_OK)
         {
         sprintf(Message, MESSAGE_4_E,
                 ERROR_IN_MQINQ, CompCode, Reason);
         SetMsg(Message);
         }
      else
         {
           /* Process the changes */
         } /* end if CompCode */
```

Setting the attributes of a queue:

This example demonstrates how to use the MQSET call to change the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CCC1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
#include <cmqc.h>      /* MQ API header file      */
⋮
#define NUMBEROFSELECTORS  2

const MQHCONN Hconn = MQHC_DEF_HCONN;

static void InhibitGetAndPut(char   *Message,
                             PMQHOBJ pHobj,
                             char   *Object)
   {
   /*                                                  */
   /*      Declare local variables                     */
   /*                                                  */
   MQLONG  SelectorCount = NUMBEROFSELECTORS;
                               /* Number of selectors  */
   MQLONG  IntAttrCount  = NUMBEROFSELECTORS;
                               /* Number of int attrs  */
   MQLONG  CharAttrLength = 0;
                    /* Length of char attribute buffer  */
   MQCHAR *CharAttrs ;
                    /* Character attribute buffer       */
   MQLONG  SelectorsTable[NUMBEROFSELECTORS];
                               /* attribute selectors  */
   MQLONG  IntAttrsTable[NUMBEROFSELECTORS];
                               /* integer attributes   */
   MQLONG  CompCode;           /* Completion code      */
   MQLONG  Reason;             /* Qualifying reason    */
⋮
   /*                                                  */
```

```
   /*     Open the queue.  If successful, do the        */
   /*     inquire call.                                 */
   /*                                                   */
.
.
.
      /*                                                */
      /*   Initialize the variables for the set call:   */
      /*    - Set SelectorsTable to the attributes to be */
      /*      set                                       */
      /*    - Set IntAttrsTable to the required status  */
      /*    - All other variables are already set       */
      /*                                                */
      SelectorsTable[0] = MQIA_INHIBIT_GET;
      SelectorsTable[1] = MQIA_INHIBIT_PUT;
      IntAttrsTable[0]  = MQQA_GET_INHIBITED;
      IntAttrsTable[1]  = MQQA_PUT_INHIBITED;
.
.
.
      /*                                                */
      /*   Issue the set call.                          */
      /*     Test the output of the set call.  If the   */
      /*     call fails, display an error message       */
      /*     showing the completion code and reason     */
      /*     code; otherwise move INHIBITED to the      */
      /*     relevant screen map fields                 */
      /*                                                */
      MQSET(Hconn,
            *pHobj,
            SelectorCount,
            SelectorsTable,
            IntAttrCount,
            IntAttrsTable,
            CharAttrLength,
            CharAttrs,
            &CompCode,
            &Reason);
      if (CompCode != MQCC_OK)
         {
         sprintf(Message, MESSAGE_4_E,
                 ERROR_IN_MQSET, CompCode, Reason);
         SetMsg(Message);
         }
      else
         {
           /* Process the changes */
         } /* end if CompCode */
```

Retrieving status information with MQSTAT:

This example demonstrates how to issue an asynchronous MQPUT and retrieve the status information with MQSTAT.

This extract is taken from the Calling MQSTAT sample application (program amqsapt0) supplied with WebSphere MQ for Windows systems. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
 /**********************************************************************/
 /*                                                                    */
 /* Program name: AMQSAPT0                                             */
 /*                                                                    */
 /* Description: Sample C program that asynchronously puts messages  */
 /*              to a message queue (example using MQPUT & MQSTAT).    */
 /*                                                                    */
 /* Licensed Materials - Property of IBM                               */
 /*                                                                    */
```

```
/* 63H9336                                                          */
/* (c) Copyright IBM Corp. 2006 All Rights Reserved.                */
/*                                                                  */
/* US Government Users Restricted Rights - Use, duplication or      */
/* disclosure restricted by GSA ADP Schedule Contract with          */
/* IBM Corp.                                                        */
/*                                                                  */
/********************************************************************/
/*                                                                  */
/* Function:                                                        */
/*                                                                  */
/*   AMQSAPT0 is a sample C program to put messages on a message    */
/*   queue with asynchronous response option, querying the success  */
/*   of the put operations with MQSTAT.                             */
/*                                                                  */
/*      -- messages are sent to the queue named by the parameter    */
/*                                                                  */
/*      -- gets lines from StdIn, and adds each to target           */
/*         queue, taking each line of text as the content           */
/*         of a datagram message; the sample stops when a null      */
/*         line (or EOF) is read.                                   */
/*         New-line characters are removed.                         */
/*         If a line is longer than 99 characters it is broken up   */
/*         into 99-character pieces. Each piece becomes the         */
/*         content of a datagram message.                           */
/*         If the length of a line is a multiple of 99 plus 1, for  */
/*         example, 199, the last piece will only contain a         */
/*         new-line character so will terminate the input.          */
/*                                                                  */
/*      -- writes a message for each MQI reason other than          */
/*         MQRC_NONE; stops if there is a MQI completion code       */
/*         of MQCC_FAILED                                           */
/*                                                                  */
/*      -- summarizes the overall success of the put operations     */
/*         through a call to MQSTAT to query MQSTAT_TYPE_ASYNC_ERROR*/
/*                                                                  */
/*    Program logic:                                                */
/*         MQOPEN target queue for OUTPUT                           */
/*         while end of input file not reached,                     */
/*         .  read next line of text                                */
/*         .  MQPUT datagram message with text line as data         */
/*         MQCLOSE target queue                                     */
/*         MQSTAT connection                                        */
/*                                                                  */
/*                                                                  */
/********************************************************************/
/*                                                                  */
/*   AMQSAPT0 has the following parameters                          */
/*       required:                                                  */
/*                 (1) The name of the target queue                 */
/*       optional:                                                  */
/*                 (2) Queue manager name                           */
/*                 (3) The open options                             */
/*                 (4) The close options                            */
/*                 (5) The name of the target queue manager         */
/*                 (6) The name of the dynamic queue                */
/*                                                                  */
/********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
   /* includes for MQI */
#include <cmqc.h>

int main(int argc, char **argv)
{
  /*  Declare file and character for sample input                  */
```

```
FILE *fp;

/*   Declare MQI structures needed                                */
MQOD     od = {MQOD_DEFAULT};    /* Object Descriptor             */
MQMD     md = {MQMD_DEFAULT};    /* Message Descriptor            */
MQPMO   pmo = {MQPMO_DEFAULT};   /* put message options           */
MQSTS   sts = {MQSTS_DEFAULT};   /* status information            */
   /** note, sample uses defaults where it can **/
MQHCONN  Hcon;                   /* connection handle             */
MQHOBJ   Hobj;                   /* object handle                 */
MQLONG   O_options;              /* MQOPEN options                */
MQLONG   C_options;              /* MQCLOSE options               */
MQLONG   CompCode;               /* completion code               */
MQLONG   OpenCode;               /* MQOPEN completion code        */
MQLONG   Reason;                 /* reason code                   */
MQLONG   CReason;                /* reason code for MQCONN        */
MQLONG   messlen;                /* message length                */
char     buffer[100];            /* message buffer                */
char     QMName[50];             /* queue manager name            */

printf("Sample AMQSAPT0 start\n");
if (argc < 2)
{
  printf("Required parameter missing - queue name\n");
  exit(99);
}

/*****************************************************************/
/*                                                               */
/*   Connect to queue manager                                    */
/*                                                               */
/*****************************************************************/
QMName[0] = 0;    /* default */
if (argc > 2)
  strcpy(QMName, argv[2]);
MQCONN(QMName,                  /* queue manager                 */
       &Hcon,                   /* connection handle             */
       &Compcode,               /* completion code               */
       &Reason);                /* reason code                   */
/* report reason and stop if it failed     */
if (CompCode == MQCC_FAILED)
{
  printf("MQCONN ended with reason code %d\n", CReason);
  exit( (int)CReason );
}

/*****************************************************************/
/*                                                               */
/*   Use parameter as the name of the target queue               */
/*                                                               */
/*****************************************************************/
strncpy(od.ObjectName, argv[1], (size_t)MQ_Q_NAME_LENGTH);
printf("target queue is %s\n", od.ObjectName);

if (argc > 5)
{
  strncpy(od.ObjectQMgrName, argv[5], (size_t) MQ_Q_MGR_NAME_LENGTH);
  printf("target queue manager is %s\n", od.ObjectQMgrName);
}

if (argc > 6)
{
  strncpy(od.DynamicQName, argv[6], (size_t) MQ_Q_NAME_LENGTH);
  printf("dynamic queue name is %s\n", od.DynamicQName);
}

/*****************************************************************/
```

```
/*                                                                  */
/*   Open the target message queue for output                       */
/*                                                                  */
/********************************************************************/
if (argc > 3)
{
  O_options = atoi( argv[3] );
  printf("open  options are %d\n", O_options);
}
else
{
  O_options = MQOO_OUTPUT            /* open queue for output     */
            | MQOO_FAIL_IF_QUIESCING /* but not if MQM stopping   */
            ;                        /* = 0x2010 = 8208 decimal   */
}

MQOPEN(Hcon,                      /* connection handle            */
       &od,                       /* object descriptor for queue  */
       O_options,                 /* open options                 */
       &Hobj,                     /* object handle                */
       &OpenCode,                 /* MQOPEN completion code       */
       &Reason);                  /* reason code                  */

/* report reason, if any; stop if failed      */
if (Reason != MQRC_NONE)
{
  printf("MQOPEN ended with reason code %d\n", Reason);
}

if (OpenCode == MQCC_FAILED)
{
  printf("unable to open queue for output\n");
}

/********************************************************************/
/*                                                                  */
/*   Read lines from the file and put them to the message queue     */
/*   Loop until null line or end of file, or there is a failure     */
/*                                                                  */
/********************************************************************/
CompCode = OpenCode;        /* use MQOPEN result for initial test */
fp = stdin;

memcpy(md.Format,           /* character string format            */
       MQFMT_STRING, (size_t)MQ_FORMAT_LENGTH);

/********************************************************************/
/* These options specify that put operation should occur            */
/* asynchronously and the application will check the success        */
/* using MQSTAT at a later time.                                    */
/********************************************************************/
md.Persistence = MQPER_NOT_PERSISTENT;
pmo.Options |= MQPMO_ASYNC_RESPONSE;

/********************************************************************/
/* These options cause the MsgId and CorrelId to be replaced, so    */
/* that there is no need to reset them before each MQPUT            */
/********************************************************************/
pmo.Options |= MQPMO_NEW_MSG_ID;
pmo.Options |= MQPMO_NEW_CORREL_ID;

while (CompCode != MQCC_FAILED)
{
  if (fgets(buffer, sizeof(buffer), fp) != NULL)
  {
    messlen = (MQLONG)strlen(buffer); /* length without null      */
    if (buffer[messlen-1] == '\n')  /* last char is a new-line    */
```

```
    {
      buffer[messlen-1]  = '\0';    /* replace new-line with null */
      --messlen;                    /* reduce buffer length       */
    }
  }
  else messlen = 0;        /* treat EOF same as null line         */

  /*************************************************************/
  /*                                                           */
  /*   Put each buffer to the message queue                    */
  /*                                                           */
  /*************************************************************/
  if (messlen > 0)
  {
    MQPUT(Hcon,               /* connection handle               */
          Hobj,               /* object handle                   */
          &md,                /* message descriptor              */
          &pmo,               /* default options (datagram)      */
          messlen,            /* message length                  */
          buffer,             /* message buffer                  */
          &CompCode,          /* completion code                 */
          &Reason);           /* reason code                     */

    /* report reason, if any */
    if (Reason != MQRC_NONE)
    {
      printf("MQPUT ended with reason code %d\n", Reason);
    }
  }
  else   /* satisfy end condition when empty line is read */
    CompCode = MQCC_FAILED;
}

/*************************************************************/
/*                                                           */
/*   Close the target queue (if it was opened)               */
/*                                                           */
/*************************************************************/
if (OpenCode != MQCC_FAILED)
{
  if (argc > 4)
  {
    C_options = atoi( argv[4] );
    printf("close options are %d\n", C_options);
  }
  else
  {
    C_options = MQCO_NONE;        /* no close options          */
  }

  MQCLOSE(Hcon,                   /* connection handle         */
          &Hobj,                  /* object handle             */
          C_options,
          &CompCode,              /* completion code           */
          &Reason);               /* reason code               */

  /* report reason, if any     */
  if (Reason != MQRC_NONE)
  {
    printf("MQCLOSE ended with reason code %d\n", Reason);
  }
}

/*************************************************************/
/*                                                           */
/*   Query how many asynchronous puts succeeded              */
/*                                                           */
```

```
  /*****************************************************************/
  MQSTAT(&Hcon,                   /* connection handle            */
         MQSTAT_TYPE_ASYNC_ERROR, /* status type                  */
         &Sts,                    /* MQSTS structure              */
         &CompCode,               /* completion code              */
         &Reason);                /* reason code                  */

  /* report reason, if any     */
  if (Reason != MQRC_NONE)
  {
    printf("MQSTAT ended with reason code %d\n", Reason);
  }
  else
  {
    /* Display results */
    printf("Succeeded putting %d messages\n",
           sts.PutSuccessCount);
    printf("%d messages were put with a warning\n",
           sts.PutWarningCount);
    printf("Failed to put %d messages\n",
           sts.PutFailureCount);

    if(sts.CompCode == MQCC_WARNING)
    {
      printf("The first warning that occurred had reason code %d\n",
             sts.Reason);
    }
    else if(sts.CompCode == MQCC_FAILED)
    {
      printf("The first error that occurred had reason code %d\n",
             sts.Reason);
    }
  }

  /*****************************************************************/
  /*                                                               */
  /*   Disconnect from MQM if not already connected                */
  /*                                                               */
  /*****************************************************************/
  if (CReason != MQRC_ALREADY_CONNECTED)
  {
    MQDISC(&Hcon,                   /* connection handle            */
           &CompCode,               /* completion code              */
           &Reason);                /* reason code                  */

    /* report reason, if any     */
    if (Reason != MQRC_NONE)
    {
      printf("MQDISC ended with reason code %d\n", Reason);
    }
  }

  /*****************************************************************/
  /*                                                               */
  /* END OF AMQSAPT0                                               */
  /*                                                               */
  /*****************************************************************/
  printf("Sample AMQSAPT0 end\n");
  return(0);
}
```

COBOL examples:

This collection of topics is taken from the WebSphere MQ for z/OS sample applications. They are applicable to all platforms, except where noted.

Connecting to a queue manager:

This example demonstrates how to use the MQCONN call to connect a program to a queue manager in z/OS batch.

This extract is taken from the Browse sample application (program CSQ4BVA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*    W02 - Data fields derived from the PARM field
 01  W02-MQM                    PIC X(48) VALUE SPACES.
*    W03 - MQM API fields
 01  W03-HCONN                  PIC S9(9) BINARY.
 01  W03-COMPCODE               PIC S9(9) BINARY.
 01  W03-REASON                 PIC S9(9) BINARY.
*
*    MQV contains constants (for filling in the control
*    blocks)
*    and return codes (for testing the result of a call)
*
 01  W05-MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
:
*    Separate into the relevant fields any data passed
*    in the PARM statement
*
     UNSTRING PARM-STRING DELIMITED BY ALL ','
                             INTO W02-MQM
                                  W02-OBJECT.
:
*    Connect to the specified queue manager.
*
     CALL 'MQCONN' USING W02-MQM
                         W03-HCONN
                         W03-COMPCODE
                         W03-REASON.
*
*    Test the output of the connect call.  If the call
*    fails, print an error message showing the
*    completion code and reason code.
*
     IF (W03-COMPCODE NOT = MQCC-OK) THEN
:
     END-IF.
:
```

Disconnecting from a queue manager:

This example demonstrates how to use the MQDISC call to disconnect a program from a queue manager in z/OS batch.

The variables used in this code extract are those that were set in "Connecting to a queue manager" on page 1357. This extract is taken from the Browse sample application (program CSQ4BVA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
⋮
*
* Disconnect from the queue manager
*
     CALL 'MQDISC' USING W03-HCONN
                         W03-COMPCODE
                         W03-REASON.
*
*    Test the output of the disconnect call.  If the
*    call fails, print an error message showing the
*    completion code and reason code.
*
     IF (W03-COMPCODE NOT = MQCC-OK) THEN
⋮
           END-IF.
⋮
```

Creating a dynamic queue:

This example demonstrates how to use the MQOPEN call to create a dynamic queue.

This extract is taken from the Credit Check sample application (program CSQ4CVB1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
⋮
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W02 - Queues processed in this program
*
 01  W02-MODEL-QNAME        PIC X(48) VALUE
     'CSQ4SAMP.B1.MODEL                          '.
 01  W02-NAME-PREFIX        PIC X(48) VALUE
     'CSQ4SAMP.B1.*                              '.
 01  W02-TEMPORARY-Q         PIC X(48).
*
*    W03 - MQM API fields
*
 01  W03-HCONN       PIC S9(9) BINARY VALUE ZERO.
 01  W03-OPTIONS     PIC S9(9) BINARY.
 01  W03-HOBJ        PIC S9(9) BINARY.
 01  W03-COMPCODE    PIC S9(9) BINARY.
 01  W03-REASON      PIC S9(9) BINARY.
*
*    API control blocks
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
*
*    CMQV contains constants (for setting or testing
```

```
*    field values) and return codes (for testing the
*    result of a call)
*
 01  MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
* ----------------------------------------------------------*
 PROCEDURE DIVISION.
* ----------------------------------------------------------*
:
* ----------------------------------------------------------*
 OPEN-TEMP-RESPONSE-QUEUE SECTION.
* ----------------------------------------------------------*
*
*  This section creates a temporary dynamic queue
*  using a model queue
*
* ----------------------------------------------------------*
*
* Change three fields in the Object Descriptor (MQOD)
* control block.  (MQODV initializes the other fields)
*
     MOVE MQOT-Q           TO MQOD-OBJECTTYPE.
     MOVE W02-MODEL-QNAME  TO MQOD-OBJECTNAME.
     MOVE W02-NAME-PREFIX  TO MQOD-DYNAMICQNAME.
*
     COMPUTE W03-OPTIONS = MQOO-INPUT-EXCLUSIVE.
*
     CALL 'MQOPEN' USING W03-HCONN
                         MQOD
                         W03-OPTIONS
                         W03-HOBJ-MODEL
                         W03-COMPCODE
                         W03-REASON.
*
     IF W03-COMPCODE NOT = MQCC-OK
          MOVE 'MQOPEN'      TO M01-MSG4-OPERATION
          MOVE W03-COMPCODE  TO M01-MSG4-COMPCODE
          MOVE W03-REASON    TO M01-MSG4-REASON
          MOVE M01-MESSAGE-4 TO M00-MESSAGE
     ELSE
          MOVE MQOD-OBJECTNAME TO W02-TEMPORARY-Q
     END-IF.
*
 OPEN-TEMP-RESPONSE-QUEUE-EXIT.
*
*    Return to performing section.
*
     EXIT.
     EJECT
*
```

Opening an existing queue:

This example demonstrates how to use the MQOPEN call to open an existing queue.

This extract is taken from the Browse sample application (program CSQ4BVA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
:
* ----------------------------------------------------------*
 WORKING-STORAGE SECTION.
* ----------------------------------------------------------*
*
```

```
*    W01 - Fields derived from the command area input
*
 01  W01-OBJECT                  PIC X(48).
*
*    W02 - MQM API fields
*
 01  W02-HCONN        PIC S9(9) BINARY VALUE ZERO.
 01  W02-OPTIONS      PIC S9(9) BINARY.
 01  W02-HOBJ         PIC S9(9) BINARY.
 01  W02-COMPCODE     PIC S9(9) BINARY.
 01  W02-REASON       PIC S9(9) BINARY.
*
*    CMQODV defines the object descriptor (MQOD)
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
*
* CMQV contains constants (for setting or testing
* field values) and return codes (for testing the
* result of a call)
*
 01  MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
* ---------------------------------------------------------*
 E-OPEN-QUEUE SECTION.
* ---------------------------------------------------------*
*                                                          *
* This section opens the queue                             *
*
*    Initialize the Object Descriptor (MQOD) control
*    block
*    (The copy file initializes the remaining fields.)
*
     MOVE MQOT-Q         TO MQOD-OBJECTTYPE.
     MOVE W01-OBJECT     TO MQOD-OBJECTNAME.
*
*    Initialize W02-OPTIONS to open the queue for both
*    inquiring about and setting attributes
*
     COMPUTE W02-OPTIONS = MQOO-INQUIRE + MQOO-SET.

*
*    Open the queue
*
     CALL 'MQOPEN' USING W02-HCONN
                         MQOD
                         W02-OPTIONS
                         W02-HOBJ
                         W02-COMPCODE
                         W02-REASON.
*
*    Test the output from the open
*
*    If the completion code is not OK, display a
*    separate error message for each of the following
*    errors:
*
*  Q-MGR-NOT-AVAILABLE - MQM is not available
*  CONNECTION-BROKEN   - MQM is no longer connected to CICS
*  UNKNOWN-OBJECT-NAME - The queue does not exist
*  NOT-AUTHORIZED      - The user is not authorized to open
*                        the queue
*
* For any other error, display an error message
* showing the completion and reason codes
*
  IF W02-COMPCODE NOT = MQCC-OK
     EVALUATE TRUE
```

```
*
       WHEN W02-REASON = MQRC-Q-MGR-NOT-AVAILABLE
            MOVE M01-MESSAGE-6 TO M00-MESSAGE
*
       WHEN W02-REASON = MQRC-CONNECTION-BROKEN
            MOVE M01-MESSAGE-6 TO M00-MESSAGE
*
       WHEN W02-REASON = MQRC-UNKNOWN-OBJECT-NAME
            MOVE M01-MESSAGE-2 TO M00-MESSAGE
*
       WHEN W02-REASON = MQRC-NOT-AUTHORIZED
            MOVE M01-MESSAGE-3 TO M00-MESSAGE
*
       WHEN OTHER
            MOVE 'MQOPEN'      TO M01-MSG4-OPERATION
            MOVE W02-COMPCODE  TO M01-MSG4-COMPCODE
            MOVE W02-REASON    TO M01-MSG4-REASON
            MOVE M01-MESSAGE-4 TO M00-MESSAGE
        END-EVALUATE
     END-IF.
 E-EXIT.
*
*    Return to performing section
*
     EXIT.
     EJECT
```

Closing a queue:

This example demonstrates how to use the MQCLOSE call.

The variables used in this code extract are those that were set in "Connecting to a queue manager" on page 1357. This extract is taken from the Browse sample application (program CSQ4BVA1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
:
*
*    Close the queue
*
     MOVE MQCO-NONE TO W03-OPTIONS.
*
     CALL 'MQCLOSE' USING W03-HCONN
                          W03-HOBJ
                          W03-OPTIONS
                          W03-COMPCODE
                          W03-REASON.
*
*    Test the output of the MQCLOSE call.  If the call
*    fails, print an error message showing the
*    completion code and reason code.
*
     IF (W03-COMPCODE NOT = MQCC-OK) THEN
        MOVE 'CLOSE'       TO W04-MSG4-TYPE
        MOVE W03-COMPCODE  TO W04-MSG4-COMPCODE
        MOVE W03-REASON    TO W04-MSG4-REASON
        MOVE W04-MESSAGE-4 TO W00-PRINT-DATA
        PERFORM PRINT-LINE
        MOVE W06-CSQ4-ERROR TO W00-RETURN-CODE
     END-IF.
*
```

Putting a message using MQPUT:

This example demonstrates how to use the MQPUT call using context.

This extract is taken from the Credit Check sample application (program CSQ4CVB1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
:
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W02 - Queues processed in this program
*
 01  W02-TEMPORARY-Q             PIC X(48).
*
*    W03 - MQM API fields
*
 01  W03-HCONN           PIC S9(9) BINARY VALUE ZERO.
 01  W03-HOBJ-INQUIRY    PIC S9(9) BINARY.
 01  W03-OPTIONS         PIC S9(9) BINARY.
 01  W03-BUFFLEN         PIC S9(9) BINARY.
 01  W03-COMPCODE        PIC S9(9) BINARY.
 01  W03-REASON          PIC S9(9) BINARY.
*
 01  W03-PUT-BUFFER.
*
     05 W03-CSQ4BIIM.
     COPY CSQ4VB1.
*
*    API control blocks
*
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-PUT-MESSAGE-OPTIONS.
     COPY CMQPMOV.
*
*    MQV contains constants (for filling in the
*    control blocks) and return codes (for testing
*    the result of a call).
*
 01  MQM-CONSTANTS.
     COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
:
*    Open queue and build message.
:
*
* Set the message descriptor and put-message options to
* the values required to create the message.
* Set the length of the message.
*
  MOVE MQMT-REQUEST         TO MQMD-MSGTYPE.
  MOVE MQCI-NONE            TO MQMD-CORRELID.
  MOVE MQMI-NONE            TO MQMD-MSGID.
  MOVE W02-TEMPORARY-Q      TO MQMD-REPLYTOQ.
  MOVE SPACES               TO MQMD-REPLYTOQMGR.
  MOVE 5                    TO MQMD-PRIORITY.
  MOVE MQPER-NOT-PERSISTENT TO MQMD-PERSISTENCE.
  COMPUTE MQPMO-OPTIONS     =  MQPMO-NO-SYNCPOINT +
                               MQPMO-DEFAULT-CONTEXT.
  MOVE LENGTH OF CSQ4BIIM-MSG TO W03-BUFFLEN.
```

```
*
     CALL 'MQPUT' USING W03-HCONN
                        W03-HOBJ-INQUIRY
                        MQMD
                        MQPMO
                        W03-BUFFLEN
                        W03-PUT-BUFFER
                        W03-COMPCODE
                        W03-REASON.
     IF W03-COMPCODE NOT = MQCC-OK
:
     END-IF.
```

Putting a message using MQPUT1:

This example demonstrates how to use the MQPUT1 call.

This extract is taken from the Credit Check sample application (program CSQ4CVB5) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
:
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W03 - MQM API fields
*
 01  W03-HCONN          PIC S9(9) BINARY VALUE ZERO.
 01  W03-OPTIONS        PIC S9(9) BINARY.
 01  W03-COMPCODE       PIC S9(9) BINARY.
 01  W03-REASON         PIC S9(9) BINARY.
 01  W03-BUFFLEN        PIC S9(9) BINARY.
*
 01  W03-PUT-BUFFER.
     05 W03-CSQ4BQRM.
     COPY CSQ4VB4.
*
*    API control blocks
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-PUT-MESSAGE-OPTIONS.
     COPY CMQPMOV.
*
* CMQV contains constants (for filling in the
* control blocks) and return codes (for testing
* the result of a call).
*
 01  MQM-MQV.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
:
*    Get the request message.
:
* -------------------------------------------------------*
 PROCESS-QUERY SECTION.
* -------------------------------------------------------*
:
*    Build the reply message.
```

```
⋮
*
* Set the object descriptor, message descriptor and
* put-message options to the values required to create
* the message.
* Set the length of the message.
*
  MOVE MQMD-REPLYTOQ    TO MQOD-OBJECTNAME.
  MOVE MQMD-REPLYTOQMGR TO MQOD-OBJECTQMGRNAME.
  MOVE MQMT-REPLY       TO MQMD-MSGTYPE.
  MOVE SPACES           TO MQMD-REPLYTOQ.
  MOVE SPACES           TO MQMD-REPLYTOQMGR.
  MOVE LOW-VALUES       TO MQMD-MSGID.
  COMPUTE MQPMO-OPTIONS = MQPMO-SYNCPOINT +
                          MQPMO-PASS-IDENTITY-CONTEXT.
  MOVE W03-HOBJ-CHECKQ  TO MQPMO-CONTEXT.
  MOVE LENGTH OF CSQ4BQRM-MSG TO W03-BUFFLEN.
*
     CALL 'MQPUT1' USING W03-HCONN
                         MQOD
                         MQMD
                         MQPMO
                         W03-BUFFLEN
                         W03-PUT-BUFFER
                         W03-COMPCODE
                         W03-REASON.
     IF W03-COMPCODE NOT = MQCC-OK
         MOVE 'MQPUT1'          TO M02-OPERATION
         MOVE MQOD-OBJECTNAME   TO M02-OBJECTNAME
         PERFORM RECORD-CALL-ERROR
         PERFORM FORWARD-MSG-TO-DLQ
     END-IF.
*
```

Getting a message:

This example demonstrates how to use the MQGET call to remove a message from a queue.

This extract is taken from the Credit Check sample application (program CSQ4CVB1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
⋮
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W03 - MQM API fields
*
 01  W03-HCONN            PIC S9(9) BINARY VALUE ZERO.
 01  W03-HOBJ-RESPONSE    PIC S9(9) BINARY.
 01  W03-OPTIONS          PIC S9(9) BINARY.
 01  W03-BUFFLEN          PIC S9(9) BINARY.
 01  W03-DATALEN          PIC S9(9) BINARY.
 01  W03-COMPCODE         PIC S9(9) BINARY.
 01  W03-REASON           PIC S9(9) BINARY.
*
 01  W03-GET-BUFFER.
     05 W03-CSQ4BAM.
     COPY CSQ4VB2.
*
*    API control blocks
*
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
```

```
 01  MQM-GET-MESSAGE-OPTIONS.
     COPY CMQGMOV.
*
*    MQV contains constants (for filling in the
*    control blocks) and return codes (for testing
*    the result of a call).
*
 01  MQM-CONSTANTS.
     COPY CMQV SUPPRESS.
* ---------------------------------------------------------*
 A-MAIN SECTION.
* ---------------------------------------------------------*
:
*    Open response queue.
:
* ---------------------------------------------------------*
 PROCESS-RESPONSE-SCREEN SECTION.
* ---------------------------------------------------------*
*                                                          *
*  This section gets a message from the response queue.    *
*                                                          *
*  When a correct response is received, it is              *
*  transferred to the map for display; otherwise           *
*  an error message is built.                              *
*                                                          *
* ---------------------------------------------------------*

*
*    Set get-message options
*
  COMPUTE MQGMO-OPTIONS = MQGMO-SYNCPOINT +
                          MQGMO-ACCEPT-TRUNCATED-MSG +
                          MQGMO-NO-WAIT.
*
* Set msgid and correlid in MQMD to nulls so that any
* message will qualify.
* Set length to available buffer length.
*
     MOVE MQMI-NONE TO MQMD-MSGID.
     MOVE MQCI-NONE TO MQMD-CORRELID.
     MOVE LENGTH OF W03-GET-BUFFER TO W03-BUFFLEN.
*
     CALL 'MQGET' USING W03-HCONN
                        W03-HOBJ-RESPONSE
                        MQMD
                        MQGMO
                        W03-BUFFLEN
                        W03-GET-BUFFER
                        W03-DATALEN
                        W03-COMPCODE
                        W03-REASON.
     EVALUATE TRUE
         WHEN W03-COMPCODE NOT = MQCC-FAILED
:
*            Process the message
:
         WHEN (W03-COMPCODE = MQCC-FAILED AND
               W03-REASON = MQRC-NO-MSG-AVAILABLE)
                 MOVE M01-MESSAGE-9 TO M00-MESSAGE
                 PERFORM CLEAR-RESPONSE-SCREEN
*
         WHEN OTHER
             MOVE 'MQGET  '     TO M01-MSG4-OPERATION
             MOVE W03-COMPCODE  TO M01-MSG4-COMPCODE
```

```
            MOVE W03-REASON    TO M01-MSG4-REASON
            MOVE M01-MESSAGE-4 TO M00-MESSAGE
            PERFORM CLEAR-RESPONSE-SCREEN
    END-EVALUATE.
```

Getting a message using the wait option:

This example demonstrates how to use the MQGET call with the wait option and accepting truncated messages.

This extract is taken from the Credit Check sample application (program CSQ4CVB5) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
:
* ------------------------------------------------------*
 WORKING-STORAGE SECTION.
* ------------------------------------------------------*
*
*    W00 - General work fields
*
 01  W00-WAIT-INTERVAL   PIC S9(09) BINARY VALUE 30000.
*
*    W03 - MQM API fields
*
 01  W03-HCONN           PIC S9(9) BINARY VALUE ZERO.
 01  W03-OPTIONS         PIC S9(9) BINARY.
 01  W03-HOBJ-CHECKQ     PIC S9(9) BINARY.
 01  W03-COMPCODE        PIC S9(9) BINARY.
 01  W03-REASON          PIC S9(9) BINARY.
 01  W03-DATALEN         PIC S9(9) BINARY.
 01  W03-BUFFLEN         PIC S9(9) BINARY.
*
 01  W03-MSG-BUFFER.
     05 W03-CSQ4BCAQ.
     COPY CSQ4VB3.
*
*    API control blocks
*
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-GET-MESSAGE-OPTIONS.
     COPY CMQGMOV.
*
*    CMQV contains constants (for filling in the
*    control blocks) and return codes (for testing
*    the result of a call).
*
 01  MQM-MQV.
 COPY CMQV SUPPRESS.
* ------------------------------------------------------*
 PROCEDURE DIVISION.
* ------------------------------------------------------*
:
*    Open input queue.
:
*
*    Get and process messages.
*
  COMPUTE MQGMO-OPTIONS = MQGMO-WAIT +
                          MQGMO-ACCEPT-TRUNCATED-MSG +
                          MQGMO-SYNCPOINT.
  MOVE LENGTH OF W03-MSG-BUFFER TO W03-BUFFLEN.
  MOVE W00-WAIT-INTERVAL TO MQGMO-WAITINTERVAL.
```

```
  MOVE MQMI-NONE TO MQMD-MSGID.
  MOVE MQCI-NONE TO MQMD-CORRELID.
*
*    Make the first MQGET call outside the loop.
*
     CALL 'MQGET' USING W03-HCONN
                        W03-HOBJ-CHECKQ
                        MQMD
                        MQGMO
                        W03-BUFFLEN
                        W03-MSG-BUFFER
                        W03-DATALEN
                        W03-COMPCODE
                        W03-REASON.
*
*    Test the output of the MQGET call using the
*    PERFORM loop that follows.
*
*    Perform whilst no failure occurs
*      - process this message
*      - reset the call parameters
*      - get another message
*    End-perform
*
:
*
*    Test the output of the MQGET call.  If the call
*    fails, send an error message showing the
*    completion code and reason code, unless the
*    completion code is NO-MSG-AVAILABLE.
*
     IF (W03-COMPCODE NOT = MQCC-FAILED) OR
        (W03-REASON NOT = MQRC-NO-MSG-AVAILABLE)
         MOVE 'MQGET '          TO M02-OPERATION
         MOVE MQOD-OBJECTNAME   TO M02-OBJECTNAME
               PERFORM RECORD-CALL-ERROR
     END-IF.
:
```

Getting a message using signaling:

This example demonstrates how to use the MQGET call with signaling. This extract is taken from the Credit Check sample application (program CSQ4CVB2) supplied with WebSphere MQ for z/OS.

Signaling is available only with WebSphere MQ for z/OS.

```
:
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W00 - General work fields
:
 01  W00-WAIT-INTERVAL    PIC S9(09) BINARY VALUE 30000.
*
*    W03 - MQM API fields
*
 01  W03-HCONN            PIC S9(9) BINARY VALUE ZERO.
 01  W03-HOBJ-REPLYQ      PIC S9(9) BINARY.
 01  W03-COMPCODE         PIC S9(9) BINARY.
 01  W03-REASON           PIC S9(9) BINARY.
 01  W03-DATALEN          PIC S9(9) BINARY.
 01  W03-BUFFLEN          PIC S9(9) BINARY.
:
 01  W03-GET-BUFFER.
```

```
     05 W03-CSQ4BQRM.
     COPY CSQ4VB4.
*
     05 W03-CSQ4BIIM REDEFINES W03-CSQ4BQRM.
     COPY CSQ4VB1.
*
     05 W03-CSQ4BPGM REDEFINES W03-CSQ4BIIM.
     COPY CSQ4VB5.
:
*    API control blocks
*
 01  MQM-MESSAGE-DESCRIPTOR.
     COPY CMQMDV.
 01  MQM-GET-MESSAGE-OPTIONS.
     COPY CMQGMOV.
:
*    MQV contains constants (for filling in the
*    control blocks) and return codes (for testing
*    the result of a call).
*
 01  MQM-MQV.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 LINKAGE SECTION.
* -------------------------------------------------------*
 01  L01-ECB-ADDR-LIST.
     05  L01-ECB-ADDR1        POINTER.
     05  L01-ECB-ADDR2        POINTER.

*
 01  L02-ECBS.
     05  L02-INQUIRY-ECB1     PIC S9(09) BINARY.
     05  L02-REPLY-ECB2       PIC S9(09) BINARY.
 01  REDEFINES L02-ECBS.
     05                       PIC  X(02).
     05  L02-INQUIRY-ECB1-CC  PIC S9(04) BINARY.
     05                       PIC  X(02).
     05  L02-REPLY-ECB2-CC    PIC S9(04) BINARY.
*
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
:
* Initialize variables, open queues, set signal on
* inquiry queue.
:
* -------------------------------------------------------*
 PROCESS-SIGNAL-ACCEPTED SECTION.
* -------------------------------------------------------*
*  This section gets a message with signal.  If a        *
*  message is received, process it.  If the signal       *
*  is set or is already set, the program goes into       *
*  an operating system wait.                             *
*  Otherwise an error is reported and call error set.    *
* -------------------------------------------------------*
*
  PERFORM REPLYQ-GETSIGNAL.
*
  EVALUATE TRUE
      WHEN (W03-COMPCODE = MQCC-OK AND
              W03-REASON = MQRC-NONE)
          PERFORM PROCESS-REPLYQ-MESSAGE
*
      WHEN (W03-COMPCODE = MQCC-WARNING AND
              W03-REASON = MQRC-SIGNAL-REQUEST-ACCEPTED)
           OR
           (W03-COMPCODE = MQCC-FAILED AND
```

```
              W03-REASON = MQRC-SIGNAL-OUTSTANDING)
           PERFORM EXTERNAL-WAIT
*
       WHEN OTHER
           MOVE 'MQGET SIGNAL'  TO M02-OPERATION
           MOVE MQOD-OBJECTNAME TO M02-OBJECTNAME
           PERFORM RECORD-CALL-ERROR
           MOVE W06-CALL-ERROR  TO W06-CALL-STATUS
  END-EVALUATE.
*
 PROCESS-SIGNAL-ACCEPTED-EXIT.
*    Return to performing section
     EXIT.
     EJECT
*
* ---------------------------------------------------------*
 EXTERNAL-WAIT SECTION.
* ---------------------------------------------------------*
*  This section performs an external CICS wait on two    *
*  ECBs until at least one is posted.  It then calls     *
*  the sections to handle the posted ECB.                *
* ---------------------------------------------------------*
     EXEC CICS WAIT EXTERNAL
         ECBLIST(W04-ECB-ADDR-LIST-PTR)
         NUMEVENTS(2)
     END-EXEC.
*
* At least one ECB must have been posted to get to this
* point.  Test which ECB has been posted and perform
* the appropriate section.
*
     IF L02-INQUIRY-ECB1 NOT = 0
         PERFORM TEST-INQUIRYQ-ECB
     ELSE
         PERFORM TEST-REPLYQ-ECB
     END-IF.
*
 EXTERNAL-WAIT-EXIT.
*
*    Return to performing section.
*
     EXIT.
     EJECT
:
* ---------------------------------------------------------*
 REPLYQ-GETSIGNAL SECTION.
* ---------------------------------------------------------*
*                                                         *
* This section performs an MQGET call (in syncpoint with *
* signal) on the reply queue.  The signal field in the   *
* MQGMO is set to the address of the ECB.                *
* Response handling is done by the performing section.   *
*                                                         *
* ---------------------------------------------------------*
*
     COMPUTE MQGMO-OPTIONS         =  MQGMO-SYNCPOINT +
                                      MQGMO-SET-SIGNAL.
     MOVE W00-WAIT-INTERVAL        TO MQGMO-WAITINTERVAL.
     MOVE LENGTH OF W03-GET-BUFFER TO W03-BUFFLEN.
*
     MOVE ZEROS                    TO L02-REPLY-ECB2.
     SET MQGMO-SIGNAL1 TO ADDRESS OF L02-REPLY-ECB2.

*
*    Set msgid and correlid to nulls so that any message
*    will qualify.
*
```

```
     MOVE MQMI-NONE TO MQMD-MSGID.
     MOVE MQCI-NONE TO MQMD-CORRELID.
*
     CALL 'MQGET' USING W03-HCONN
                        W03-HOBJ-REPLYQ
                        MQMD
                        MQGMO
                        W03-BUFFLEN
                        W03-GET-BUFFER
                        W03-DATALEN
                        W03-COMPCODE
                        W03-REASON.
*
 REPLYQ-GETSIGNAL-EXIT.
*
*    Return to performing section.
*
     EXIT.
     EJECT
*
.
.
.
```

Inquiring about the attributes of an object:

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CVC1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS).

```
.
.
.
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W02 - MQM API fields
*
 01  W02-SELECTORCOUNT    PIC S9(9) BINARY VALUE 2.
 01  W02-INTATTRCOUNT     PIC S9(9) BINARY VALUE 2.
 01  W02-CHARATTRLENGTH   PIC S9(9) BINARY VALUE ZERO.
 01  W02-CHARATTRS        PIC X     VALUE LOW-VALUES.
 01  W02-HCONN            PIC S9(9) BINARY VALUE ZERO.
 01  W02-HOBJ             PIC S9(9) BINARY.
 01  W02-COMPCODE         PIC S9(9) BINARY.
 01  W02-REASON           PIC S9(9) BINARY.
 01  W02-SELECTORS-TABLE.
     05  W02-SELECTORS    PIC S9(9) BINARY OCCURS 2 TIMES
 01  W02-INTATTRS-TABLE.
     05  W02-INTATTRS     PIC S9(9) BINARY OCCURS 2 TIMES
*
*    CMQODV defines the object descriptor (MQOD).
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
*
* CMQV contains constants (for setting or testing field
* values) and return codes (for testing the result of a
* call).
*
 01  MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
*
```

```
*    Get the queue name and open the queue.
*
:
*
*    Initialize the variables for the inquiry call:
*     - Set W02-SELECTORS-TABLE to the attributes whose
*    status is required
*     - All other variables are already set
*
     MOVE MQIA-INHIBIT-GET TO W02-SELECTORS(1).
     MOVE MQIA-INHIBIT-PUT TO W02-SELECTORS(2).
*
*    Inquire about the attributes.
*
     CALL 'MQINQ' USING W02-HCONN,
                        W02-HOBJ,
                        W02-SELECTORCOUNT,
                        W02-SELECTORS-TABLE,
                        W02-INTATTRCOUNT,
                        W02-INTATTRS-TABLE,
                        W02-CHARATTRLENGTH,
                        W02-CHARATTRS,
                        W02-COMPCODE,
                        W02-REASON.
*
* Test the output from the inquiry:
*
* - If the completion code is not OK, display an error
*   message showing the completion and reason codes
*
* - Otherwise, move the correct attribute status into
*   the relevant screen map fields
*
     IF W02-COMPCODE NOT = MQCC-OK
        MOVE 'MQINQ'       TO M01-MSG4-OPERATION
        MOVE W02-COMPCODE  TO M01-MSG4-COMPCODE
        MOVE W02-REASON    TO M01-MSG4-REASON
        MOVE M01-MESSAGE-4 TO M00-MESSAGE
*
     ELSE
*       Process the changes.
:
           END-IF.
:
```

Setting the attributes of a queue:

This example demonstrates how to use the MQSET call to change the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CVC1) supplied with WebSphere MQ for z/OS. For the names and locations of the sample applications on other platforms, see Sample programs (platforms except z/OS)

```
:
* -------------------------------------------------------*
 WORKING-STORAGE SECTION.
* -------------------------------------------------------*
*
*    W02 - MQM API fields
*
 01  W02-SELECTORCOUNT    PIC S9(9) BINARY VALUE 2.
 01  W02-INTATTRCOUNT     PIC S9(9) BINARY VALUE 2.
 01  W02-CHARATTRLENGTH   PIC S9(9) BINARY VALUE ZERO.
 01  W02-CHARATTRS        PIC X     VALUE LOW-VALUES.
```

```
 01  W02-HCONN            PIC S9(9) BINARY VALUE ZERO.
 01  W02-HOBJ             PIC S9(9) BINARY.
 01  W02-COMPCODE         PIC S9(9) BINARY.
 01  W02-REASON           PIC S9(9) BINARY.
 01  W02-SELECTORS-TABLE.
     05  W02-SELECTORS    PIC S9(9) BINARY OCCURS 2 TIMES.
 01  W02-INTATTRS-TABLE.
     05  W02-INTATTRS     PIC S9(9) BINARY OCCURS 2 TIMES.
*
*    CMQODV defines the object descriptor (MQOD).
*
 01  MQM-OBJECT-DESCRIPTOR.
     COPY CMQODV.
*
* CMQV contains constants (for setting or testing
* field values) and return codes (for testing the
* result of a call).
*
 01  MQM-CONSTANTS.
 COPY CMQV SUPPRESS.
* -------------------------------------------------------*
 PROCEDURE DIVISION.
* -------------------------------------------------------*
*
*    Get the queue name and open the queue.
*
:
*
*
* Initialize the variables required for the set call:
* - Set W02-SELECTORS-TABLE to the attributes to be set
* - Set W02-INTATTRS-TABLE  to the required status
* - All other variables are already set
*
     MOVE MQIA-INHIBIT-GET   TO W02-SELECTORS(1).
     MOVE MQIA-INHIBIT-PUT   TO W02-SELECTORS(2).
     MOVE MQQA-GET-INHIBITED TO W02-INTATTRS(1).
     MOVE MQQA-PUT-INHIBITED TO W02-INTATTRS(2).
*
*    Set the attributes.
*
     CALL 'MQSET' USING W02-HCONN,
                        W02-HOBJ,
                        W02-SELECTORCOUNT,
                        W02-SELECTORS-TABLE,
                        W02-INTATTRCOUNT,
                        W02-INTATTRS-TABLE,
                        W02-CHARATTRLENGTH,
                        W02-CHARATTRS,
                        W02-COMPCODE,
                        W02-REASON.
*
* Test the output from the call:
*
*  - If the completion code is not OK, display an error
*    message showing the completion and reason codes
*
*  - Otherwise, move 'INHIBITED' into the relevant
*    screen map fields
*
     IF W02-COMPCODE NOT = MQCC-OK
        MOVE 'MQSET'       TO M01-MSG4-OPERATION
        MOVE W02-COMPCODE  TO M01-MSG4-COMPCODE
        MOVE W02-REASON    TO M01-MSG4-REASON
        MOVE M01-MESSAGE-4 TO M00-MESSAGE
     ELSE
*
```

```
*       Process the changes.
⋮
     END-IF.
```

System/390 assembler-language examples:

This collection of topics is mostly taken from the WebSphere MQ for z/OS sample applications.

Connecting to a queue manager:

This example demonstrates how to use the MQCONN call to connect a program to a queue manager in z/OS batch.

This extract is taken from the Browse sample program (CSQ4BAA1) supplied with WebSphere MQ for z/OS.

```
⋮
WORKAREA DSECT
*
PARMLIST CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
COMPCODE DS    F              Completion code
REASON   DS    F              Reason code
HCONN    DS    F              Connection handle
         ORG
PARMADDR DS    F              Address of parm field
PARMLEN  DS    H              Length of parm field
*
MQMNAME  DS    CL48           Queue manager name
*
*
************************************************************
*  SECTION NAME : MAINPARM                                 *
************************************************************
MAINPARM DS    0H
         MVI   MQMNAME,X'40'
         MVC   MQMNAME+1(L'MQMNAME-1),MQMNAME
*
* Space out first byte and initialize
*
*
* Code to address and verify parameters passed omitted
*
*
PARM1MVE DS    0H
         SR    R1,R3           Length of data
         LA    R4,MQMNAME      Address for target
         BCTR  R1,R0           Reduce for execute
         EX    R1,MOVEPARM     Move the data
*
************************************************************
* EXECUTES                                                 *
************************************************************
MOVEPARM MVC   0(*-*,R4),0(R3)
*
         EJECT
************************************************************
*  SECTION NAME : MAINCONN                                 *
************************************************************
*
*
```

```
MAINCONN DS    0H
         XC    HCONN,HCONN       Null connection handle
*
         CALL  MQCONN,                          X
               (MQMNAME,                        X
               HCONN,                           X
               COMPCODE,                        X
               REASON),                         X
               MF=(E,PARMLIST),VL
*
         LA    R0,MQCC_OK        Expected compcode
         C     R0,COMPCODE       As expected?
         BER   R6                Yes .. return to caller
*
         MVC   INF4_TYP,=CL10'CONNECT   '
         BAL   R7,ERRCODE              Translate error
         LA    R0,8                    Set exit code
         ST    R0,EXITCODE             to 8
         B     ENDPROG                 End the program
*
```

Disconnecting from a queue manager:

This example demonstrates how to use the MQDISC call to disconnect a program from a queue manager in z/OS batch.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```
.
.
.
*
*        ISSUE MQI DISC REQUEST USING REENTRANT FORM
*        OF CALL MACRO
*
*        HCONN WAS SET BY A PREVIOUS MQCONN REQUEST
*        R5 = WORK REGISTER
*
DISC     DS    0H
         CALL  MQDISC,                       X
               (HCONN,                       X
               COMPCODE,                     X
               REASON),                      X
               VL,MF=(E,CALLLST)
*
         LA    R5,MQCC_OK
         C     R5,COMPCODE
         BNE   BADCALL
.
.
.
BADCALL  DS    0H
.
.
.
*                   CONSTANTS
*
         CMQA
*
*        WORKING STORAGE (RE-ENTRANT)
*
WEG3     DSECT
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
HCONN    DS   F
COMPCODE DS   F
REASON   DS   F
```

```
*
*
LEG3     EQU  *-WKEG3
         END
```

Creating a dynamic queue:

This example demonstrates how to use the MQOPEN call to create a dynamic queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```
:
*
*    R5 = WORK REGISTER.
*
OPEN     DS   0H
*
         MVC  WOD_AREA,MQOD_AREA INITIALIZE WORKING VERSION OF
*                                     MQOD WITH DEFAULTS
         MVC  WOD_OBJECTNAME,MOD_Q    COPY IN THE MODEL Q NAME
         MVC  WOD_DYNAMICQNAME,DYN_Q  COPY IN THE DYNAMIC Q NAME
         L    R5,=AL4(MQOO_OUTPUT)    OPEN FOR OUTPUT AND
         A    R5,=AL4(MQOO_INQUIRE)   INQUIRE
         ST   R5,OPTIONS
*
* ISSUE MQI OPEN REQUEST USING REENTRANT
* FORM OF CALL MACRO
*
         CALL MQOPEN,                            X
               (HCONN,                           X
               WOD,                              X
               OPTIONS,                          X
               HOBJ,                             X
               COMPCODE,                         X
               REASON),VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK                CHECK THE COMPLETION CODE
         C   R5,COMPCODE               FROM THE REQUEST AND BRANCH
         BNE BADCALL                   TO ERROR ROUTINE IF NOT MQCC_OK
*
         MVC  TEMP_Q,WOD_OBJECTNAME  SAVE NAME OF TEMPORARY Q
*                             CREATED BY OPEN OF MODEL Q
*
:
BADCALL  DS   0H
:
*
*
*   CONSTANTS:
*
MOD_Q  DC   CL48'QUERY.REPLY.MODEL'  MODEL QUEUE NAME
DYN_Q  DC   CL48'QUERY.TEMPQ.*'      DYNAMIC QUEUE NAME
*
       CMQODA DSECT=NO,LIST=YES  CONSTANT VERSION OF MQOD
       CMQA                      MQI VALUE EQUATES
*
*   WORKING STORAGE
*
         DFHEISTG
HCONN    DS F                      CONNECTION HANDLE
OPTIONS  DS F                      OPEN OPTIONS
HOBJ     DS F                      OBJECT HANDLE
COMPCODE DS F                      MQI COMPLETION CODE
REASON   DS F                      MQI REASON CODE
TEMP_Q   DS CL(MQ_Q_NAME_LENGTH)   SAVED QNAME AFTER OPEN
```

```
*
WOD      CMQODA DSECT=NO,LIST=YES  WORKING VERSION OF MQOD
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0,0),VL,MF=L  LIST FORM
                                                   OF CALL
*                                                   MACRO
.
.
.
         END
```

Opening an existing queue:

This example demonstrates how to use the MQOPEN call to open a queue that has already been defined.

It shows how to specify two options. This extract is not taken from the sample applications supplied with WebSphere MQ.

```
.
.
.
*
*    R5 = WORK REGISTER.
*
OPEN     DS   0H
*
         MVC  WOD_AREA,MQOD_AREA  INITIALIZE WORKING VERSION OF
*                              MQOD WITH DEFAULTS
         MVC  WOD_OBJECTNAME,Q_NAME    SPECIFY Q NAME TO OPEN
         LA   R5,MQOO_INPUT_EXCLUSIVE  OPEN FOR MQGET CALLS
*
         ST   R5,OPTIONS
*
* ISSUE MQI OPEN REQUEST USING REENTRANT FORM
* OF CALL MACRO
*
         CALL MQOPEN,                          X
               (HCONN,                         X
               WOD,                            X
               OPTIONS,                        X
               HOBJ,                           X
               COMPCODE,                       X
               REASON),VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK         CHECK THE COMPLETION CODE
         C   R5,COMPCODE        FROM THE REQUEST AND BRANCH
         BNE BADCALL            TO ERROR ROUTINE IF NOT MQCC_OK
*
.
.
.
BADCALL  DS   0H
.
.
.
*
*
*   CONSTANTS:
*
Q_NAME   DC   CL48'REQUEST.QUEUE'  NAME OF QUEUE TO OPEN
*
         CMQODA DSECT=NO,LIST=YES  CONSTANT VERSION OF MQOD
         CMQA                      MQI VALUE EQUATES
*
*   WORKING STORAGE
*
         DFHEISTG
HCONN    DS F           CONNECTION HANDLE
OPTIONS  DS F           OPEN OPTIONS
HOBJ     DS F           OBJECT HANDLE
COMPCODE DS F           MQI COMPLETION CODE
REASON   DS F           MQI REASON CODE
*
```

```
WOD  CMQODA DSECT=NO,LIST=YES  WORKING VERSION OF MQOD
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L  LIST FORM
                                                 OF CALL
*                                                 MACRO
:
         END
```

Closing a queue:

This example demonstrates how to use the MQCLOSE call to close a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```
:
*
* ISSUE MQI CLOSE REQUEST USING REENTRANT FROM OF
* CALL MACRO
*
*        HCONN WAS SET BY A PREVIOUS MQCONN REQUEST
*        HOBJ  WAS SET BY A PREVIOUS MQOPEN REQUEST
*        R5 = WORK REGISTER
*
CLOSE    DS    0H
         LA    R5,MQCO_NONE        NO SPECIAL CLOSE OPTIONS
         ST    R5,OPTIONS              ARE REQUIRED.
*
         CALL  MQCLOSE,                       X
               (HCONN,                        X
               HOBJ,                          X
               OPTIONS,                       X
               COMPCODE,                      X
               REASON),                       X
               VL,MF=(E,CALLLST)
*
         LA    R5,MQCC_OK
         C     R5,COMPCODE
         BNE   BADCALL
*
:
BADCALL  DS    0H
:
*                   CONSTANTS
*
         CMQA
*
*        WORKING STORAGE (REENTRANT)
*
WEG4     DSECT
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
HCONN    DS   F
HOBJ     DS   F
OPTIONS  DS   F
COMPCODE DS   F
REASON   DS   F
*
*
LEG4     EQU  *-WKEG4
         END
```

Putting a message using MQPUT:

This example demonstrates how to use the MQPUT call to put a message on a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```
:
*     CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
:
*
*     OPEN A QUEUE
*
OPEN    DS  0H
:
*
*     R4,R5,R6,R7 = WORK REGISTER.
*
PUT  DS  0H
     LA   R4,MQMD           SET UP ADDRESSES AND
     LA   R5,MQMD_LENGTH    LENGTH FOR USE BY MVCL
     LA   R6,WMD            INSTRUCTION, AS MQMD IS
     LA   R7,WMD_LENGTH     OVER 256 BYES LONG.
     MVCL R6,R4             INITIALIZE WORKING VERSION
*                           OF MESSAGE DESCRIPTOR
*
     MVC  WPMO_AREA,MQPMO_AREA  INITIALIZE WORKING MQPMO
*

     LA   R5,BUFFER_LEN   RETRIEVE THE BUFFER LENGTH
     ST   R5,BUFFLEN      AND SAVE IT FOR MQM USE
*
     MVC  BUFFER,TEST_MSG    SET THE MESSAGE TO BE PUT
*
*    ISSUE MQI PUT REQUEST USING REENTRANT FORM
*    OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
         CALL MQPUT,                          X
               (HCONN,                        X
               HOBJ,                          X
               WMD,                           X
               WPMO,                          X
               BUFFLEN,                       X
               BUFFER,                        X
               COMPCODE,                      X
               REASON),VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK
         C   R5,COMPCODE
         BNE BADCALL
*
:
BADCALL DS  0H
:
*
*      CONSTANTS
*
   CMQMDA DSECT=NO,LIST=YES,PERSISTENCE=MQPER_PERSISTENT
   CMQPMOA DSECT=NO,LIST=YES
   CMQA
TEST_MSG DC CL80'THIS IS A TEST MESSAGE'
```

```
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO,LIST=NO
WPMO     CMQPMOA DSECT=NO,LIST=NO
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
:
         END
```

Putting a message using MQPUT1:

This example demonstrates how to use the MQPUT1 call to open a queue, put a single message on the queue, then close the queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```
:
*
*     CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
:
*
*     R4,R5,R6,R7 = WORK REGISTER.
*
PUT      DS  0H
*
         MVC  WOD_AREA,MQOD_AREA     INITIALIZE WORKING VERSION OF
*                                    MQOD WITH DEFAULTS
         MVC  WOD_OBJECTNAME,Q_NAME  SPECIFY Q NAME FOR PUT1
*
         LA   R4,MQMD           SET UP ADDRESSES AND
         LA   R5,MQMD_LENGTH    LENGTH FOR USE BY MVCL
         LA   R6,WMD            INSTRUCTION, AS MQMD IS
         LA   R7,WMD_LENGTH     OVER 256 BYES LONG.
         MVCL R6,R4             INITIALIZE WORKING VERSION
*                        OF MESSAGE DESCRIPTOR
*
         MVC  WPMO_AREA,MQPMO_AREA       INITIALIZE WORKING MQPMO
*

         LA   R5,BUFFER_LEN           RETRIEVE THE BUFFER LENGTH
         ST   R5,BUFFLEN              AND SAVE IT FOR MQM USE
*
         MVC  BUFFER,TEST_MSG         SET THE MESSAGE TO BE PUT
*
* ISSUE MQI PUT REQUEST USING REENTRANT FORM OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
```

```
*
         CALL MQPUT1,                                  X
               (HCONN,                                 X
                LMQOD,                                 X
                LMQMD,                                 X
                LMQPMO,                                X
                BUFFERLENGTH,                          X
                BUFFER,                                X
                COMPCODE,                              X
                REASON),VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK
         C   R5,COMPCODE
         BNE BADCALL
*
:
BADCALL DS  0H
:
*
*      CONSTANTS
*
  CMQMDA DSECT=NO,LIST=YES,PERSISTENCE=MQPER_PERSISTENT
  CMQPMOA DSECT=NO,LIST=YES
  CMQODA DSECT=NO,LIST=YES
  CMQA
*
TEST_MSG DC CL80'THIS IS ANOTHER TEST MESSAGE'
Q_NAME   DC CL48'TEST.QUEUE.NAME'
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WOD      CMQODA DSECT=NO,LIST=YES    WORKING VERSION OF MQOD
WMD      CMQMDA DSECT=NO,LIST=NO
WPMO     CMQPMOA DSECT=NO,LIST=NO
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
:
         END
```

Getting a message:

This example demonstrates how to use the MQGET call to remove a message from a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```
:
*
*     CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
```

```
:
*
*     OPEN A QUEUE FOR GET
*
OPEN    DS  0H
:
*
*     R4,R5,R6,R7 = WORK REGISTER.
*
GET  DS  0H
     LA   R4,MQMD                SET UP ADDRESSES AND
     LA   R5,MQMD_LENGTH         LENGTH FOR USE BY MVCL
     LA   R6,WMD                 INSTRUCTION, AS MQMD IS
     LA   R7,WMD_LENGTH          OVER 256 BYES LONG.
     MVCL R6,R4                  INITIALIZE WORKING VERSION
*                                   OF MESSAGE DESCRIPTOR
*
     MVC  WGMO_AREA,MQGMO_AREA   INITIALIZE WORKING MQGMO
*

     LA   R5,BUFFER_LEN          RETRIEVE THE BUFFER LENGTH
     ST   R5,BUFFLEN             AND SAVE IT FOR MQM USE
*
*
* ISSUE MQI GET REQUEST USING REENTRANT FORM OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
         CALL  MQGET,                           X
               (HCONN,                          X
               HOBJ,                            X
               WMD,                             X
               WGMO,                            X
               BUFFLEN,                         X
               BUFFER,                          X
               DATALEN,                         X
               COMPCODE,                        X
               REASON),                         X
               VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK
         C   R5,COMPCODE
         BNE BADCALL
*
:
BADCALL DS  0H
:
*
*      CONSTANTS
*
         CMQMDA DSECT=NO,LIST=YES
         CMQGMOA DSECT=NO,LIST=YES
         CMQA
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
```

```
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO,LIST=NO
WGMO     CMQGMOA DSECT=NO,LIST=NO
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
.
.
.
         END
```

Getting a message using the wait option:

This example demonstrates how to use the wait option of the MQGET call.

This code accepts truncated messages. This extract is not taken from the sample applications supplied with WebSphere MQ.

```
.
.
.
*     CONNECT TO QUEUE MANAGER
CONN    DS  0H
.
.
.
*     OPEN A QUEUE FOR GET
OPEN    DS  0H
.
.
.
*     R4,R5,R6,R7 = WORK REGISTER.
GET  DS  0H
     LA   R4,MQMD              SET UP ADDRESSES AND
     LA   R5,MQMD_LENGTH       LENGTH FOR USE BY MVCL
     LA   R6,WMD               INSTRUCTION, AS MQMD IS
     LA   R7,WMD_LENGTH        OVER 256 BYES LONG.
     MVCL R6,R4                INITIALIZE WORKING VERSION
*                                  OF MESSAGE DESCRIPTOR
*
     MVC  WGMO_AREA,MQGMO_AREA   INITIALIZE WORKING MQGMO
     L    R5,=AL4(MQGMO_WAIT)
     A    R5,=AL4(MQGMO_ACCEPT_TRUNCATED_MSG)
     ST   R5,WGMO_OPTIONS
     MVC  WGMO_WAITINTERVAL,TWO_MINUTES   WAIT UP TO TWO
                                          MINUTES BEFORE
                                          FAILING THE
                                          CALL
*
     LA   R5,BUFFER_LEN     RETRIEVE THE BUFFER LENGTH
     ST   R5,BUFFLEN        AND SAVE IT FOR MQM USE
*
*  ISSUE MQI GET REQUEST USING REENTRANT FORM OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
         CALL  MQGET,                           X
               (HCONN,                          X
               HOBJ,                            X
               WMD,                             X
               WGMO,                            X
               BUFFLEN,                         X
               BUFFER,                          X
               DATALEN,                         X
               COMPCODE,                        X
               REASON),                         X
               VL,MF=(E,CALLLST)
*
```

```
  LA  R5,MQCC_OK                  DID THE MQGET REQUEST
  C   R5,COMPCODE                   WORK OK?
  BE  GETOK                       YES, SO GO AND PROCESS.
  LA  R5,MQCC_WARNING             NO, SO CHECK FOR A WARNING.
  C   R5,COMPCODE                 IS THIS A WARNING?
  BE  CHECK_W                     YES, SO CHECK THE REASON.
*
  LA  R5,MQRC_NO_MSG_AVAILABLE   IT MUST BE AN ERROR.
                                 IS IT DUE TO AN EMPTY
  C   R5,REASON                  QUEUE?
  BE  NOMSG                      YES, SO HANDLE THE ERROR
  B   BADCALL                    NO, SO GO TO ERROR ROUTINE
*
CHECK_W  DS  0H
         LA  R5,MQRC_TRUNCATED_MSG_ACCEPTED   IS THIS A
                                              TRUNCATED
         C   R5,REASON                        MESSAGE?
         BE  GETOK                   YES, SO GO AND PROCESS.
         B   BADCALL                 NO, SOME OTHER WARNING
*
NOMSG    DS  0H
:
GETOK    DS  0H
:
BADCALL DS  0H
:
*
*      CONSTANTS
*
         CMQMDA DSECT=NO,LIST=YES
         CMQGMOA DSECT=NO,LIST=YES
         CMQA
*
TWO_MINUTES DC F'120000'       GET WAIT INTERVAL
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO,LIST=NO
WGMO     CMQGMOA DSECT=NO,LIST=NO
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
*
:
         END
```

Getting a message using signaling:

This example demonstrates how to use the MQGET call to set a signal so that you are notified when a suitable message arrives on a queue.

This extract is not taken from the sample applications supplied with WebSphere MQ.

```
:
*
*     CONNECT TO QUEUE MANAGER
*
CONN    DS  0H
:
*
*     OPEN A QUEUE FOR GET
*
OPEN    DS  0H
:
*
*     R4,R5,R6,R7 = WORK REGISTER.
*
GET  DS  0H
     LA   R4,MQMD                SET UP ADDRESSES AND
     LA   R5,MQMD_LENGTH         LENGTH FOR USE BY MVCL
     LA   R6,WMD                 INSTRUCTION, AS MQMD IS
     LA   R7,WMD_LENGTH          OVER 256 BYES LONG.
     MVCL R6,R4                  INITIALIZE WORKING VERSION
*                                   OF MESSAGE DESCRIPTOR
*
     MVC  WGMO_AREA,MQGMO_AREA   INITIALIZE WORKING MQGMO
     LA   R5,MQGMO_SET_SIGNAL
     ST   R5,WGMO_OPTIONS
     MVC  WGMO_WAITINTERVAL,FIVE_MINUTES   WAIT UP TO FIVE
                                           MINUTES BEFORE
*                                          FAILING THE CALL
*
     XC   SIG_ECB,SIG_ECB   CLEAR THE ECB
     LA   R5,SIG_ECB        GET THE ADDRESS OF THE ECB
     ST   R5,WGMO_SIGNAL1   AND PUT IT IN THE WORKING
*                                MQGMO
*

     LA   R5,BUFFER_LEN     RETRIEVE THE BUFFER LENGTH
     ST   R5,BUFFLEN        AND SAVE IT FOR MQM USE
*
*
*    ISSUE MQI GET REQUEST USING REENTRANT FORM OF CALL MACRO
*
*        HCONN WAS SET BY PREVIOUS MQCONN REQUEST
*        HOBJ WAS SET BY PREVIOUS MQOPEN REQUEST
*
         CALL  MQGET,                          X
               (HCONN,                         X
               HOBJ,                           X
               WMD,                            X
               WGMO,                           X
               BUFFLEN,                        X
               BUFFER,                         X
               DATALEN,                        X
               COMPCODE,                       X
               REASON),                        X
               VL,MF=(E,CALLLST)
*
         LA  R5,MQCC_OK        DID THE MQGET REQUEST
         C   R5,COMPCODE       WORK OK?
```

```
         BE  GETOK             YES, SO GO AND PROCESS.
         LA  R5,MQCC_WARNING   NO, SO CHECK FOR A WARNING.
         C   R5,COMPCODE       IS THIS A WARNING?
         BE  CHECK_W           YES, SO CHECK THE REASON.
         B   BADCALL           NO, SO GO TO ERROR ROUTINE
*
CHECK_W  DS  0H
         LA  R5,MQRC_SIGNAL_REQUEST_ACCEPTED
         C   R5,REASON   SIGNAL REQUEST SIGNAL SET?
         BNE BADCALL     NO, SOME ERROR OCCURRED
         B   DOWORK      YES, SO DO SOMETHING
*                          ELSE
*
CHECKSIG DS  0H
         CLC SIG_ECB+1(3),=AL3(MQEC_MSG_ARRIVED)
                             IS A MESSAGE AVAILABLE?
         BE  GET               YES, SO GO AND GET IT
*
         CLC SIG_ECB+1(3),=AL3(MQEC_WAIT_INTERVAL_EXPIRED)
                           HAVE WE WAITED LONG ENOUGH?
         BE  NOMSG         YES, SO SAY NO MSG AVAILABLE
         B   BADCALL       IF IT'S ANYTHING ELSE
*                          GO TO ERROR ROUTINE.
*
DOWORK   DS  0H
:
         TM  SIG_ECB,X'40'   HAS THE SIGNAL ECB BEEN POSTED?
         BO  CHECKSIG        YES, SO GO AND CHECK WHY
         B   DOWORK          NO, SO GO AND DO MORE WORK
*
NOMSG    DS  0H
:
GETOK    DS  0H
:
BADCALL DS  0H
:
*
*      CONSTANTS
*
         CMQMDA DSECT=NO,LIST=YES
         CMQGMOA DSECT=NO,LIST=YES
         CMQA
*
FIVE_MINUTES DC F'300000'        GET SIGNAL INTERVAL
*
*      WORKING STORAGE DSECT
*
WORKSTG  DSECT
*
COMPCODE DS F
REASON   DS F
BUFFLEN  DS F
DATALEN  DS F
OPTIONS  DS F
HCONN    DS F
HOBJ     DS F
SIG_ECB  DS F

*
BUFFER   DS CL80
BUFFER_LEN EQU *-BUFFER
*
WMD      CMQMDA DSECT=NO,LIST=NO
WGMO     CMQGMOA DSECT=NO,LIST=NO
*
CALLLST  CALL ,(0,0,0,0,0,0,0,0,0,0,0),VL,MF=L
```

```
*
.
.
.
        END
```

Inquiring about and setting the attributes of a queue:

This example demonstrates how to use the MQINQ call to inquire about the attributes of a queue and to use the MQSET call to change the attributes of a queue.

This extract is taken from the Queue Attributes sample application (program CSQ4CAC1) supplied with WebSphere MQ for z/OS.

```
.
.
.
DFHEISTG DSECT
.
.
.
OBJDESC  CMQODA LIST=YES   Working object descriptor
*
SELECTORCOUNT   DS F       Number of selectors
INTATTRCOUNT    DS F       Number of integer attributes
CHARATTRLENGTH  DS F       char attributes length
CHARATTRS       DS C       Area for char attributes
*
OPTIONS  DS   F            Command options
HCONN    DS   F            Handle of connection
HOBJ     DS   F            Handle of object
COMPCODE DS   F            Completion code
REASON   DS   F            Reason code
SELECTOR DS   2F           Array of selectors
INTATTRS DS   2F           Array of integer attributes
.
.
.
OBJECT   DS   CL(MQ_Q_NAME_LENGTH)   Name of queue
.
.
.
CALLLIST CALL ,(0,0,0,0,0,0,0,0,0,0),VL,MF=L
************************************************************
*                   PROGRAM EXECUTION STARTS HERE        *
.
.
.
CSQ4CAC1 DFHEIENT CODEREG=(R3),DATAREG=(R13)
.
.
.
*        Initialize the variables for the set call
*
         SR   R0,R0               Clear register zero
         ST   R0,CHARATTRLENGTH   Set char length to zero
         LA   R0,2                Load to set
         ST   R0,SELECTORCOUNT    selectors add
         ST   R0,INTATTRCOUNT     integer attributes
*
         LA   R0,MQIA_INHIBIT_GET Load q attribute selector
         ST   R0,SELECTOR+0       Place in field
         LA   R0,MQIA_INHIBIT_PUT Load q attribute selector
         ST   R0,SELECTOR+4       Place in field
*
UPDTEST  DS   0H
         CLC  ACTION,CINHIB       Are we inhibiting?
         BE   UPDINHBT            Yes branch to section
*
         CLC  ACTION,CALLOW       Are we allowing?
         BE   UPDALLOW            Yes branch to section
*
         MVC  M00_MSG,M01_MSG1    Invalid request
         BR   R6                  Return to caller
*
```

```
UPDINHBT DS   0H
         MVC  UPDTYPE,CINHIBIT      Indicate action type
         LA   R0,MQQA_GET_INHIBITED Load attribute value
         ST   R0,INTATTRS+0         Place in field
         LA   R0,MQQA_PUT_INHIBITED Load attribute value
         ST   R0,INTATTRS+4         Place in field
         B    UPDCALL               Go and do call
*
UPDALLOW DS   0H
         MVC  UPDTYPE,CALLOWED      Indicate action type
         LA   R0,MQQA_GET_ALLOWED   Load attribute value
         ST   R0,INTATTRS+0         Place in field
         LA   R0,MQQA_PUT_ALLOWED   Load attribute value
         ST   R0,INTATTRS+4         Place in field
         B    UPDCALL               Go and do call
*
UPDCALL  DS   0H
         CALL MQSET,                          C
               (HCONN,                        C
               HOBJ,                          C
               SELECTORCOUNT,                 C
               SELECTOR,                      C
               INTATTRCOUNT,                  C
               INTATTRS,                      C
               CHARATTRLENGTH,                C
               CHARATTRS,                     C
               COMPCODE,                      C
               REASON),                       C
               VL,MF=(E,CALLLIST)
*
         LA   R0,MQCC_OK    Load expected compcode
         C    R0,COMPCODE   Was set successful?
.
.
.
*  SECTION NAME : INQUIRE                                  *
*  FUNCTION     : Inquires on the objects attributes       *
*  CALLED BY    : PROCESS                                  *
*  CALLS        : OPEN, CLOSE, CODES                       *
*  RETURN       : To Register 6                            *
INQUIRE  DS   0H
.
.
.

*        Initialize the variables for the inquire call
*
         SR   R0,R0                Clear register zero
         ST   R0,CHARATTRLENGTH    Set char length to zero
         LA   R0,2                 Load to set
         ST   R0,SELECTORCOUNT     selectors add
         ST   R0,INTATTRCOUNT      integer attributes
*
         LA   R0,MQIA_INHIBIT_GET Load attribute value
         ST   R0,SELECTOR+0        Place in field
         LA   R0,MQIA_INHIBIT_PUT Load attribute value
         ST   R0,SELECTOR+4        Place in field
         CALL MQINQ,                          C
               (HCONN,                        C
               HOBJ,                          C
               SELECTORCOUNT,                 C
               SELECTOR,                      C
               INTATTRCOUNT,                  C
               INTATTRS,                      C
               CHARATTRLENGTH,                C
               CHARATTRS,                     C
               COMPCODE,                      C
               REASON),                       C
               VL,MF=(E,CALLLIST)
         LA   R0,MQCC_OK        Load expected compcode
```

```
        C    R0,COMPCODE       Was inquire successful?
⋮
```

Constants

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

WebSphere MQ COPY, header, include, and module files:

This information is general-use programming interface information.

This section contains information to help you use the MQI for various programming languages, as follows.

C header files:

Header files are provided to help you write C application programs that use the MQI. These header files are summarized in the table:

Table 110. C header files - call prototypes, data types, return codes, constants, and structures

File name	Description	IBM i	UNIX and Linux systems	Windows	z/OS
Call prototypes, data types, return codes, constants, and structures					
CMQC	MQI definitions	C	C	C	C
CMQBC	MQAI definitions	C	C	C	
CMQEC	Interface Entry Points definition (includes CMQC, CMQXC and CMQZC)		C	C	
CMQCFC	PCF definitions	C	C	C	C
CMQPSC	Publish/Subscribe definitions	C	C	C	C
CMQXC	Channel and exit definitions	C	C	C	C
CMQZC	Installable services definitions	C	C	C	
Key: C= Files provided					

COBOL COPY files:

Various COPY files are provided to help you write COBOL application programs that use the MQI. These files are summarized in the table:

Table 111. COBOL copy files - return codes, constants, and structures

File name	Description	IBM i	UNIX systems	Windows	z/OS
Return codes and constants					
CMQx	MQI definitions	V	V	V	V
CMQCFx	PCF definitions	V	V	V	V
CMQPSx	Publish/Subscribe definitions	V	V	V	V
CMQXx	Channel and exit definitions	V	V	V	V
Structures					

Table 111. COBOL copy files - return codes, constants, and structures (continued)

File name	Description	IBM i		UNIX systems		Windows		z/OS	
CMQAIRx	MQAIR - Authentication information record			V	L	V	L		
CMQBOx	MQBO - Begin options	V	L	V	L	V	L		
CMQCDx	MQCD - Channel defintion	V	L	V	L	V	L	V	L
CMQCFBFx	MQCFBF - PCF byte string filter parameter	V	L	V	L	V	L	V	L
CMQCFBSx	MQCFBS - PCF byte string parameter	V	L	V	L	V	L	V	L
CMQCFGRx	MQCFGR - PCF group parameter	V	L	V	L	V	L	V	L
CMQCFHx	MQCFH - PCF header	V	L	V	L	V	L	V	L
CMQCFIFx	MQCFIF - PCF integer filter parameter	V	L	V	L	V	L	V	L
CMQCFILx	MQCFIL - PCF integer list parameter	V	L	V	L	V	L	V	L
CMQCFINx	MQCFIN - PCF integer parameter	V	L	V	L	V	L	V	L
CMQCFSFx	MQCFSF - PCF string filter parameter	V	L	V	L	V	L	V	L
CMQCFSLx	MQCFSL - PCF string list parameter	V	L	V	L	V	L	V	L
CMQCFSTx	MQCFST - PCF string parameter	V	L	V	L	V	L	V	L
CMQCFXLx	MQCFIL64 - PCF 64-bit integer list parameter	V	L	V	L	V	L	V	L
CMQCFXNx	MQCFIN64 - PCF 64-bit integer parameter	V	L	V	L	V	L	V	L
CMQCHRVx	MQCHARV - Variable length string	V	L	V	L	V	L	V	L
CMQCIHx	MQCIH - CICS bridge header	V	L	V	L	V	L	V	L
CMQCNOx	MQCNO - Connect options	V	L	V	L	V	L	V	L
CMQCSPx	MQCSP - Security parameters	V	L	V	L	V	L	V	L
CMQCXPx	MQCXP - Channel exit parameters	V	L					V	L
CMQDHx	MQDH - Distribution header	V	L	V	L	V	L	V	L
CMQDLHx	MQDLH - Dead-letter header	V	L	V	L	V	L	V	L
CMQDXPx	MQDXP - Data conversion exit parameters	V	L			V	L		
CMQEPHx	MQEPH - Embedded PCF header	V	L	V	L	V	L	V	L
CMQGMOx	MQGMO - Get message options	V	L	V	L	V	L	V	L
CMQIIHx	MQIIH - IMS information header	V	L	V	L	V	L	V	L
CMQMDx	MQMD - Message descriptor	V	L	V	L	V	L	V	L
CMQMD1x	MQMD1 - Message descriptor version 1	V	L	V	L	V	L	V	L
CMQMD2x	MQMD2 - Message descriptor version 2	V	L	V	L	V	L	V	L
CMQMDEx	MQMDE - Message descriptor extended	V	L	V	L	V	L	V	L
CMQODx	MQOD - Object descriptor	V	L	V	L	V	L	V	L

Table 111. COBOL copy files - return codes, constants, and structures (continued)

File name	Description	IBM i	UNIX systems	Windows	z/OS
CMQORx	MQOR - Object record	V L	V L	V L	V L
CMQPMOx	MQPMO - Put message options	V L	V L	V L	V L
CMQRFHx	MQRFH - Rules and formatting header	V L	V L	V L	V L
CMQRFH2x	MQRFH2 - Rules and formatting header 2	V L	V L	V L	V L
CMQRMHx	MQRMH - Reference message header	V L	V L	V L	V L
CMQRRx	MQRR - Response record	V L	V L	V L	
CMQSCOx	MQSCO - SSL configuraton options		V L	V L	
CMQTMx	MQTM - Trigger message	V L		V L	V L
CMQTMCx	MQTMC - Trigger message character	V L	V L		
CMQTMC2x	MQTMC2 - Trigger message 2 character	V L	V L	V L	V L
CMQWIHx	MQWIH - Work information header	V L	V L	V L	V L
CMQXQHx	MQXQH - Transmission queue header	V L	V L	V L	V L

Key:

- Files with initial values provided, x=V
- Files without initial values provided, x=L

PL/I include files:

The following INCLUDE files are provided for the PL/I programming language. These files are available on z/OS only.

Table 112. PL/I include files - data types, return codes, constants, and structures

File name	Description	IBM i	UNIX systems	Windows	z/OS
Data types, return codes, constants, and structures					
CMQP	MQI definitions				P
CMQCFP	PCF definitions				P
CMQEPP	Entry point definitions				P
CMQPSP	Publish/Subscribe definitions				P
CMQXP	Channel and exit definitions				P

Key: P= File provided

RPG copy files:

The following COPY files are provided for the RPG programming language. These files are available only on IBM i.

Table 113. RPG copy files - return codes, constants, and structures

File name	Description	IBM i	UNIX systems	Windows	z/OS
Return codes and constants					
CMQx	MQI definitions	G R			
CMQCFx	PCF definitions	G			
CMQPSx	Publish/Subscribe definitions	G			
CMQXx	Channel and exit definitions	G R			
Structures					
CMQBOx	MQBO - Begin options	G H			
CMQCDx	MQCD - Channel definition	G H R			
CMQCFBFx	MQCFBF - PCF byte string filter parameter	G H			
CMQCFBSx	MQCFBS - PCF byte string parameter	G H			
CMQCFGRx	MQCFGR - PCF group parameter	G H			
CMQCFHx	MQCFH - PCF header	G H			
CMQCFIFx	MQCFIF - PCF integer filter parameter	G H			
CMQCFILx	MQCFIL - PCF integer list parameter	G H			
CMQCFINx	MQCFIN - PCF integer parameter	G H			
CMQCFSFx	MQCFSF - PCF string filter parameter	G H			
CMQCFSLx	MQCFSL - PCF string list parameter	G H			
CMQCFSTx	MQCFST - PCF string parameter	G H			
CMQCFXLx	MQCFIL64 - PCF 64-bit integer list parameter	G H			
CMQCFXNx	MQCFIN64 - PCF 64-bit integer parameter	G H			
CMQCHARVx	MQCHARV - Variable length string	G H			
CMQCIHx	MQCIH - CICS bridge header	G H			
CMQCNOx	MQCNO - Connect options	G H			
CMQCSPx	MQCSP - Security parameters	G H			
CMQCXPx	MQCXP - Channel exit parameters	G H R			
CMQDHx	MQDH - Distribution header	G H R			
CMQDLHx	MQDLH - Dead-letter header	G H R			
CMQDXPx	MQDXP - Data conversion exit parameters	G H R			
CMQEPHx	MQEPH - Embedded PCF header	G H			
CMQGMOx	MQGMO - Get message options	G H R			
CMQIIHx	MQIIH - IMS information header	G H R			
CMQMDx	MQMD - Message descriptor	G H R			
CMQMD1x	MQMD1 - Message descriptor version 1	G H R			

Table 113. RPG copy files - return codes, constants, and structures (continued)

File name	Description	IBM i	UNIX systems	Windows	z/OS
CMQMD2x	MQMD2 - Message descriptor version 2	G H			
CMQMDEx	MQMDE - Message descriptor extended	G H R			
CMQODx	MQOD - Object descriptor	G H R			
CMQORx	MQOR - Object record	G H R			
CMQPMOx	MQPMO - Put message options	G H R			
CMQPXPx	MQPXP - Publish/Subscribe routing exit parameters	G H			
CMQRFHx	MQRFH - Rules and formatting header	G H			
CMQRFH2x	MQRFH2 - Rules and formatting header 2	G H			
CMQRMHx	MQRMH - Reference message header	G H R			
CMQRRx	MQRR - Response record	G H R			
CMQTMx	MQTM - Trigger message	G H R			
CMQTMCx	MQTMC - Trigger message character	G H R			
CMQTMC2x	MQTMC2 - Trigger message 2 character	G H R			
CMQWIHx	MQWIH - Work information header	G H			
CMQXQHx	MQXQH - Transmission queue header	G H R			

Key:

- File for static linkage, initialized, provided x=G
- File for static linkage, not initialized, provided x=H
- File for dynamic linkage, initialized, provided, x=R

Visual Basic module files:

Header (or form) files are provided to help you write Visual Basic application programs that use the MQI. These header files are supplied in 32-bit versions only and are summarized in the table:

Table 114. Visual Basic module files - call declarations, data types, return codes, constants, and structures

File name	Description	IBM i	UNIX and Linux systems	Windows	z/OS
Call declarations, data types, return codes, constants, and structures					
CMQB	MQI definitions			B	
CMQBB	MQAI definitions			B	
CMQCFB	PCF definitions			B	
CMQXB	Channel and exit definitions			B	

Key: B= File provided

Constants:

List of all the constants

MQ_* (String Lengths):

MQ_ABEND_CODE_LENGTH	4	X'00000004'
MQ_ACCOUNTING_TOKEN_LENGTH	32	X'00000020'
MQ_APPL_IDENTITY_DATA_LENGTH	32	X'00000020'
MQ_APPL_NAME_LENGTH	28	X'0000001C'
MQ_APPL_ORIGIN_DATA_LENGTH	4	X'00000004'
MQ_APPL_TAG_LENGTH	28	X'0000001C'
MQ_ARM_SUFFIX_LENGTH	2	X'00000002'
MQ_ATTENTION_ID_LENGTH	4	X'00000004'
MQ_AUTH_INFO_CONN_NAME_LENGTH	264	X'00000108'
MQ_AUTH_INFO_DESC_LENGTH	64	X'00000040'
MQ_AUTH_INFO_NAME_LENGTH	48	X'00000030'
MQ_AUTH_INFO_OCSP_URL_LENGTH	256	X'00000100'
MQ_AUTHENTICATOR_LENGTH	8	X'00000008'
MQ_AUTO_REORG_CATALOG_LENGTH	44	X'0000002C'
MQ_AUTO_REORG_TIME_LENGTH	4	X'00000004'
MQ_BATCH_INTERFACE_ID_LENGTH	8	X'00000008'
MQ_BRIDGE_NAME_LENGTH	24	X'00000018'
MQ_CANCEL_CODE_LENGTH	4	X'00000004'
MQ_CF_STRUC_DESC_LENGTH	64	X'00000040'
MQ_CF_STRUC_NAME_LENGTH	12	X'0000000C'
MQ_CHANNEL_DATE_LENGTH	12	X'0000000C'
MQ_CHANNEL_DESC_LENGTH	64	X'00000040'
MQ_CHANNEL_NAME_LENGTH	20	X'00000014'
MQ_CHANNEL_TIME_LENGTH	8	X'00000008'
MQ_CHINIT_SERVICE_PARM_LENGTH	32	X'00000020'
MQ_CICS_FILE_NAME_LENGTH	8	X'00000008'
MQ_CLIENT_ID_LENGTH	23	X'00000017'
MQ_CLUSTER_NAME_LENGTH	48	X'00000030'
MQ_CONN_NAME_LENGTH	264	X'00000108'
MQ_CONN_TAG_LENGTH	128	X'00000080'
MQ_CONNECTION_ID_LENGTH	24	X'00000018'
MQ_CORREL_ID_LENGTH	24	X'00000018'
MQ_CREATION_DATE_LENGTH	12	X'0000000C'
MQ_CREATION_TIME_LENGTH	8	X'00000008'
MQ_DATE_LENGTH	12	X'0000000C'
MQ_DISTINGUISHED_NAME_LENGTH	1024	X'00000400'
MQ_DNS_GROUP_NAME_LENGTH	18	X'00000012'
MQ_EXIT_DATA_LENGTH	32	X'00000020'

MQ_EXIT_INFO_NAME_LENGTH	48	X'00000030'
MQ_EXIT_NAME_LENGTH	(value differs by platform or version)	
MQ_EXIT_PD_AREA_LENGTH	48	X'00000030'
MQ_EXIT_USER_AREA_LENGTH	16	X'00000010'
MQ_FACILITY_LENGTH	8	X'00000008'
MQ_FACILITY_LIKE_LENGTH	4	X'00000004'
MQ_FORMAT_LENGTH	8	X'00000008'
MQ_FUNCTION_LENGTH	4	X'00000004'
MQ_GROUP_ID_LENGTH	24	X'00000018'
MQ_LDAP_PASSWORD_LENGTH	32	X'00000020'
MQ_LISTENER_NAME_LENGTH	48	X'00000030'
MQ_LISTENER_DESC_LENGTH	64	X'00000040'
MQ_LOCAL_ADDRESS_LENGTH	48	X'00000030'
MQ_LTERM_OVERRIDE_LENGTH	8	X'00000008'
MQ_LU_NAME_LENGTH	8	X'00000008'
MQ_LUWID_LENGTH	16	X'00000010'
MQ_MAX_EXIT_NAME_LENGTH	128	X'00000080'
MQ_MAX_MCA_USER_ID_LENGTH	64	X'00000040'
MQ_MAX_PROPERTY_NAME_LENGTH	4095	X'00000FFF'
MQ_MAX_USER_ID_LENGTH	64	X'00000040'
MQ_MCA_JOB_NAME_LENGTH	28	X'0000001C'
MQ_MCA_NAME_LENGTH	20	X'00000014'
MQ_MCA_USER_DATA_LENGTH	32	X'00000020'
MQ_MCA_USER_ID_LENGTH	(value differs by platform or version)	
MQ_MFS_MAP_NAME_LENGTH	8	X'00000008'
MQ_MODE_NAME_LENGTH	8	X'00000008'
MQ_MSG_HEADER_LENGTH	4000	X'00000FA0'
MQ_MSG_ID_LENGTH	24	X'00000018'
MQ_MSG_TOKEN_LENGTH	16	X'00000010'
MQ_NAMELIST_DESC_LENGTH	64	X'00000040'
MQ_NAMELIST_NAME_LENGTH	48	X'00000030'
MQ_OBJECT_INSTANCE_ID_LENGTH	24	X'00000018'
MQ_OBJECT_NAME_LENGTH	48	X'00000030'
MQ_PASS_TICKET_APPL_LENGTH	8	X'00000008'
MQ_PASSWORD_LENGTH	12	X'0000000C'
MQ_PROCESS_APPL_ID_LENGTH	256	X'00000100'
MQ_PROCESS_DESC_LENGTH	64	X'00000040'
MQ_PROCESS_ENV_DATA_LENGTH	128	X'00000080'
MQ_PROCESS_NAME_LENGTH	48	X'00000030'
MQ_PROCESS_USER_DATA_LENGTH	128	X'00000080'

MQ_PROGRAM_NAME_LENGTH	20	X'00000014'
MQ_PUT_APPL_NAME_LENGTH	28	X'0000001C'
MQ_PUT_DATE_LENGTH	8	X'00000008'
MQ_PUT_TIME_LENGTH	8	X'00000008'
MQ_Q_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_IDENTIFIER_LENGTH	48	X'00000030'
MQ_Q_MGR_NAME_LENGTH	48	X'00000030'
MQ_Q_NAME_LENGTH	48	X'00000030'
MQ_QSG_NAME_LENGTH	4	X'00000004'
MQ_REMOTE_SYS_ID_LENGTH	4	X'00000004'
MQ_SECURITY_ID_LENGTH	40	X'00000028'
MQ_SELECTOR_LENGTH	10240	X'00002800'
MQ_SERVICE_ARGS_LENGTH	255	X'000000FF'
MQ_SERVICE_COMMAND_LENGTH	255	X'000000FF'
MQ_SERVICE_DESC_LENGTH	64	X'00000040'
MQ_SERVICE_NAME_LENGTH	32	X'00000020'
MQ_SERVICE_PATH_LENGTH	255	X'000000FF'
MQ_SERVICE_STEP_LENGTH	8	X'00000008'
MQ_SHORT_CONN_NAME_LENGTH	20	X'00000014'
MQ_SHORT_DNAME_LENGTH	256	X'00000100'
MQ_SSL_CIPHER_SPEC_LENGTH	32	X'00000020'
MQ_SSL_CRYPTO_HARDWARE_LENGTH	256	X'00000100'
MQ_SSL_HANDSHAKE_STAGE_LENGTH	32	X'00000020'
MQ_SSL_KEY_LIBRARY_LENGTH	44	X'0000002C'
MQ_SSL_KEY_MEMBER_LENGTH	8	X'00000008'
MQ_SSL_KEY_REPOSITORY_LENGTH	256	X'00000100'
MQ_SSL_PEER_NAME_LENGTH	1024	X'00000400'
MQ_SSL_SHORT_PEER_NAME_LENGTH	256	X'00000100'
MQ_START_CODE_LENGTH	4	X'00000004'
MQ_STORAGE_CLASS_DESC_LENGTH	64	X'00000040'
MQ_STORAGE_CLASS_LENGTH	8	X'00000008'
MQ_SUB_IDENTITY_LENGTH	128	X'00000080'
MQ_SUB_POINT_LENGTH	128	X'00000080'
MQ_SUITE_B_128_BIT	2	X'00000002'
MQ_SUITE_B_192_BIT	4	X'00000004'
MQ_SUITE_B_NONE	1	X'00000001'
MQ_SUITE_B_NOT_AVAILABLE	0	X'00000000'
MQ_TCP_NAME_LENGTH	8	X'00000008'
MQ_TIME_LENGTH	8	X'00000008'
MQ_TOPIC_DESC_LENGTH	64	X'00000040'
MQ_TOPIC_NAME_LENGTH	48	X'00000030'

MQ_TOPIC_STR_LENGTH	10240	X'00002800'
MQ_TOTAL_EXIT_DATA_LENGTH	999	X'000003E7'
MQ_TOTAL_EXIT_NAME_LENGTH	999	X'000003E7'
MQ_TP_NAME_LENGTH	64	X'00000040'
MQ_TPIPE_NAME_LENGTH	8	X'00000008'
MQ_TRAN_INSTANCE_ID_LENGTH	16	X'00000010'
MQ_TRANSACTION_ID_LENGTH	4	X'00000004'
MQ_TRIGGER_DATA_LENGTH	64	X'00000040'
MQ_TRIGGER_PROGRAM_NAME_LENGTH	8	X'00000008'
MQ_TRIGGER_TERM_ID_LENGTH	4	X'00000004'
MQ_TRIGGER_TRANS_ID_LENGTH	4	X'00000004'
MQ_USER_ID_LENGTH	12	X'0000000C'
MQ_VERSION_LENGTH	8	X'00000008'
MQ_XCF_GROUP_NAME_LENGTH	8	X'00000008'
MQ_XCF_MEMBER_NAME_LENGTH	16	X'00000010'

MQ_* (Command format String Lengths):

MQ_ARCHIVE_PFX_LENGTH	36	X'00000024'
MQ_ARCHIVE_UNIT_LENGTH	8	X'00000008'
MQ_ASID_LENGTH	4	X'00000004'
MQ_AUTH_PROFILE_NAME_LENGTH	48	X'00000030'
MQ_CF_LEID_LENGTH	12	X'0000000C'
MQ_COMMAND_MQSC_LENGTH	32768	X'00008000'
MQ_DATA_SET_NAME_LENGTH	44	X'0000002C'
MQ_DB2_NAME_LENGTH	4	X'00000004'
MQ_DSG_NAME_LENGTH	8	X'00000008'
MQ_ENTITY_NAME_LENGTH	64	X'00000040'
MQ_ENV_INFO_LENGTH	96	X'00000060'
MQ_IP_ADDRESS_LENGTH	48	X'00000030'
MQ_LOG_CORREL_ID_LENGTH	8	X'00000008'
MQ_LOG_EXTENT_NAME_LENGTH	24	X'00000018'
MQ_LOG_PATH_LENGTH	1024	X'00000400'
MQ_LRSN_LENGTH	12	X'0000000C'
MQ_ORIGIN_NAME_LENGTH	8	X'00000008'
MQ_PSB_NAME_LENGTH	8	X'00000008'
MQ_PST_ID_LENGTH	8	X'00000008'
MQ_Q_MGR_CPF_LENGTH	4	X'00000004'
MQ_RESPONSE_ID_LENGTH	24	X'00000018'
MQ_RBA_LENGTH	12	X'0000000C'
MQ_SECURITY_PROFILE_LENGTH	40	X'00000028'
MQ_SERVICE_COMPONENT_LENGTH	48	X'00000030'

MQ_SUB_NAME_LENGTH	10240	X'00002800'
MQ_SYSP_SERVICE_LENGTH	32	X'00000020'
MQ_SYSTEM_NAME_LENGTH	8	X'00000008'
MQ_TASK_NUMBER_LENGTH	8	X'00000008'
MQ_TPIPE_PFX_LENGTH	4	X'00000004'
MQ_UOW_ID_LENGTH	256	X'00000100'
MQ_USER_DATA_LENGTH	10240	X'00002800'
MQ_VOLSER_LENGTH	6	X'00000006'

MQACH_* (API exit chain area header structure):

MQACH_STRUC_ID	"ACHb"	
MQACH_STRUC_ID_ARRAY	'A','C','H','b'	
MQACH_VERSION_1	1	X'00000001'
MQACH_CURRENT_VERSION	1	X'00000001'
MQACH_LENGTH_1	(value differs by platform or version)	
MQACH_CURRENT_LENGTH	(value differs by platform or version)	

MQACT_* (Accounting Token):

MQACT_NONE	X'00...00'	(32 nulls)
MQACT_NONE_ARRAY	'\0','\0',...	(32 nulls)

MQACT_* (Command format Action Options)

MQACT_FORCE_REMOVE	1	X'00000001'
MQACT_ADVANCE_LOG	2	X'00000002'
MQACT_COLLECT_STATISTICS	3	X'00000003'
MQACT_PUBSUB	4	X'00000004'

MQACTP_* (Action):

MQACTP_NEW	0	X'00000000'
MQACTP_FORWARD	1	X'00000001'
MQACTP_REPLY	2	X'00000002'
MQACTP_REPORT	3	X'00000003'

MQACTT_* (Accounting Token Types):

MQACTT_UNKNOWN	X'00'	
MQACTT_CICS_LUOW_ID	X'01'	
MQACTT_OS2_DEFAULT	X'04'	
MQACTT_DOS_DEFAULT	X'05'	
MQACTT_UNIX_NUMERIC_ID	X'06'	
MQACTT_OS400_ACCOUNT_TOKEN	X'08'	
MQACTT_WINDOWS_DEFAULT	X'09'	
MQACTT_NT_SECURITY_ID	X'0B'	
MQACTT_USER	X'19'	

MQADOPT_* (Adopt New MCA Checks and Adopt New MCA Types):
Adopt New MCA Checks

MQADOPT_CHECK_NONE	0	X'00000000'
MQADOPT_CHECK_ALL	1	X'00000001'
MQADOPT_CHECK_Q_MGR_NAME	2	X'00000002'
MQADOPT_CHECK_NET_ADDR	4	X'00000004'

Adopt New MCA Types

MQADOPT_TYPE_NO	0	X'00000000'
MQADOPT_TYPE_ALL	1	X'00000001'
MQADOPT_TYPE_SVR	2	X'00000002'
MQADOPT_TYPE_SDR	4	X'00000004'
MQADOPT_TYPE_RCVR	8	X'00000008'
MQADOPT_TYPE_CLUSRCVR	16	X'00000010'

MQAIR_* (Authentication information record structure):

MQAIR_STRUC_ID	"AIRb"	
MQAIR_STRUC_ID_ARRAY	'A','I','R','b'	
MQAIR_VERSION_1	1	X'00000001'
MQAIR_VERSION_2	2	X'00000002'
MQAIR_CURRENT_VERSION	2	X'00000002'

MQAIT_* (Authentication Information Type):

MQAIT_ALL	0	X'00000000'
MQAIT_CRL_LDAP	1	X'00000001'
MQAIT_OCSP	2	X'00000002'

MQAS_* (Command format Asynchronous State Values):

MQAS_NONE	0	X'00000000'
MQAS_STARTED	1	X'00000001'
MQAS_START_WAIT	2	X'00000002'
MQAS_STOPPED	3	X'00000003'
MQAS_SUSPENDED	4	X'00000004'
MQAS_SUSPENDED_TEMPORARY	5	X'00000005'
MQAS_ACTIVE	6	X'00000006'
MQAS_INACTIVE	7	X'00000007'

MQAT_* (Put Application Types):

MQAT_UNKNOWN	-1	X'FFFFFFFF'
MQAT_NO_CONTEXT	0	X'00000000'
MQAT_CICS	1	X'00000001'
MQAT_MVS	2	X'00000002'
MQAT_OS390	2	X'00000002'
MQAT_ZOS	2	X'00000002'
MQAT_IMS	3	X'00000003'
MQAT_OS2	4	X'00000004'
MQAT_DOS	5	X'00000005'
MQAT_AIX	6	X'00000006'
MQAT_UNIX	6	X'00000006'
MQAT_QMGR	7	X'00000007'
MQAT_OS400	8	X'00000008'
MQAT_WINDOWS	9	X'00000009'
MQAT_CICS_VSE	10	X'0000000A'
MQAT_WINDOWS_NT	11	X'0000000B'
MQAT_VMS	12	X'0000000C'
MQAT_GUARDIAN	13	X'0000000D'
MQAT_NSK	13	X'0000000D'
MQAT_VOS	14	X'0000000E'
MQAT_OPEN_TP1	15	X'0000000F'
MQAT_VM	18	X'00000012'
MQAT_IMS_BRIDGE	19	X'00000013'
MQAT_XCF	20	X'00000014'
MQAT_CICS_BRIDGE	21	X'00000015'
MQAT_NOTES_AGENT	22	X'00000016'

MQAT_TPF	23	X'00000017'
MQAT_USER	25	X'00000019'
MQAT_BROKER	26	X'0000001A'
MQAT_QMGR_PUBLISH	26	X'0000001A'
MQAT_JAVA	28	X'0000001C'
MQAT_DQM	29	X'0000001D'
MQAT_CHANNEL_INITIATOR	30	X'0000001E'
MQAT_WLM	31	X'0000001F'
MQAT_BATCH	32	X'00000020'
MQAT_RRS_BATCH	33	X'00000021'
MQAT_SIB	34	X'00000022'
MQAT_DEFAULT	(value differs by platform or version)	
MQAT_USER_FIRST	65536	X'00010000'
MQAT_USER_LAST	999999999	X'3B9AC9FF'

MQAUTH_* (Command format Authority Values):

MQAUTH_NONE	0	X'00000000'
MQAUTH_ALT_USER_AUTHORITY	1	X'00000001'
MQAUTH_BROWSE	2	X'00000002'
MQAUTH_CHANGE	3	X'00000003'
MQAUTH_CLEAR	4	X'00000004'
MQAUTH_CONNECT	5	X'00000005'
MQAUTH_CREATE	6	X'00000006'
MQAUTH_DELETE	7	X'00000007'
MQAUTH_DISPLAY	8	X'00000008'
MQAUTH_INPUT	9	X'00000009'
MQAUTH_INQUIRE	10	X'0000000A'
MQAUTH_OUTPUT	11	X'0000000B'
MQAUTH_PASS_ALL_CONTEXT	12	X'0000000C'
MQAUTH_PASS_IDENTITY_CONTEXT	13	X'0000000D'
MQAUTH_SET	14	X'0000000E'
MQAUTH_SET_ALL_CONTEXT	15	X'0000000F'
MQAUTH_SET_IDENTITY_CONTEXT	16	X'00000010'
MQAUTH_CONTROL	17	X'00000011'
MQAUTH_CONTROL_EXTENDED	18	X'00000012'
MQAUTH_PUBLISH	19	X'00000013'
MQAUTH_SUBSCRIBE	20	X'00000014'
MQAUTH_RESUME	21	X'00000015'
MQAUTH_SYSTEM	22	X'00000016'

MQAUTHOPT_* (Command format Authority Options):

MQAUTHOPT_CUMULATIVE	256	X'00000100'
MQAUTHOPT_ENTITY_EXPLICIT	1	X'00000001'
MQAUTHOPT_ENTITY_SET	2	X'00000002'
MQAUTHOPT_NAME_ALL_MATCHING	32	X'00000020'
MQAUTHOPT_NAME_AS_WILDCARD	64	X'00000040'
MQAUTHOPT_NAME_EXPLICIT	16	X'00000010'

MQAXC_* (API exit context structure):

MQAXC_STRUC_ID	"AXCb"	
MQAXC_STRUC_ID_ARRAY	'A','X','C','b'	
MQAXC_VERSION_1	1	X'00000001'
MQAXC_CURRENT_VERSION	1	X'00000001'

MQAXP_* (API exit parameter structure):

MQAXP_STRUC_ID	"AXPb"	
MQAXP_STRUC_ID_ARRAY	'A','X','P','b'	
MQAXP_VERSION_1	1	X'00000001'
MQAXP_VERSION_2	2	X'00000002'
MQAXP_CURRENT_VERSION	2	X'00000002'

MQBA_* (Byte Attribute Selectors):

MQBA_FIRST	6001	X'00001771'
MQBA_LAST	8000	X'00001F40'

MQBACF_* (Command format Byte Parameter Types):

MQBACF_FIRST	7001	X'00001B59'
MQBACF_EVENT_ACCOUNTING_TOKEN	7001	X'00001B59'
MQBACF_EVENT_SECURITY_ID	7002	X'00001B5A'
MQBACF_RESPONSE_SET	7003	X'00001B5B'
MQBACF_RESPONSE_ID	7004	X'00001B5C'
MQBACF_EXTERNAL_UOW_ID	7005	X'00001B5D'
MQBACF_CONNECTION_ID	7006	X'00001B5E'
MQBACF_GENERIC_CONNECTION_ID	7007	X'00001B5F'
MQBACF_ORIGIN_UOW_ID	7008	X'00001B60'
MQBACF_Q_MGR_UOW_ID	7009	X'00001B61'
MQBACF_ACCOUNTING_TOKEN	7010	X'00001B62'
MQBACF_CORREL_ID	7011	X'00001B63'
MQBACF_GROUP_ID	7012	X'00001B64'
MQBACF_MSG_ID	7013	X'00001B65'
MQBACF_CF_LEID	7014	X'00001B66'

MQBACF_DESTINATION_CORREL_ID	7015	X'00001B67'
MQBACF_SUB_ID	7016	X'00001B68'
MQBACF_LAST_USED	7016	X'00001B68'

MQBL_* (Buffer Length for mqAddString and mqSetString):

MQBL_NULL_TERMINATED	-1	X'FFFFFFFF'

MQBMHO_* (Buffer to message handle options and structure):
Buffer to message handle options structure

MQBMHO_STRUC_ID	"BMHO"	
MQBMHO_STRUC_ID_ARRAY	'B','M','H','O'	
MQBMHO_VERSION_1	1	X'00000001'
MQBMHO_CURRENT_VERSION	1	X'00000001'

Buffer To Message Handle Options

MQBMHO_NONE	0	X'00000000'
MQBMHO_DELETE_PROPERTIES	1	X'00000001'

MQBND_* (Default Bindings):

MQBND_BIND_ON_OPEN	0	X'00000000'
MQBND_BIND_NOT_FIXED	1	X'00000001'
MQBND_BIND_ON_GROUP	2	X'00000002'

MQBO_* (Begin options and structure):
Begin options structure

MQBO_STRUC_ID	"BObb"	
MQBO_STRUC_ID_ARRAY	'B','O','b','b'	
MQBO_VERSION_1	1	X'00000001'
MQBO_CURRENT_VERSION	1	X'00000001'

Begin Options

MQBO_NONE	0	X'00000000'

MQBT_* (Command format Bridge Types):

MQBT_OTMA	1	X'00000001'

MQCA_* (Character Attribute Selectors):

MQCA_ADMIN_TOPIC_NAME	2105	X'00000839'
MQCA_ALTERATION_DATE	2027	X'000007EB'
MQCA_ALTERATION_TIME	2028	X'000007EC'
MQCA_APPL_ID	2001	X'000007D1'
MQCA_AUTH_INFO_CONN_NAME	2053	X'00000805'
MQCA_AUTH_INFO_DESC	2046	X'000007FE'
MQCA_AUTH_INFO_NAME	2045	X'000007FD'
MQCA_AUTH_INFO_OCSP_URL	2109	X'0000083D'
MQCA_AUTO_REORG_CATALOG	2091	X'0000082B'
MQCA_AUTO_REORG_START_TIME	2090	X'0000082A'
MQCA_BACKOUT_REQ_Q_NAME	2019	X'000007E3'
MQCA_BASE_OBJECT_NAME	2002	X'000007D2'
MQCA_BASE_Q_NAME	2002	X'000007D2'
MQCA_BATCH_INTERFACE_ID	2068	X'00000814'
MQCA_CF_STRUC_DESC	2052	X'00000804'
MQCA_CF_STRUC_NAME	2039	X'000007F7'
MQCA_CHANNEL_AUTO_DEF_EXIT	2026	X'000007EA'
MQCA_CHILD	2101	X'00000835'
MQCA_CHINIT_SERVICE_PARM	2076	X'0000081C'
MQCA_CICS_FILE_NAME	2060	X'0000080C'
MQCA_CLUS_CHL_NAME	2124	X'0000084C'
MQCA_CLUSTER_DATE	2037	X'000007F5'
MQCA_CLUSTER_NAME	2029	X'000007ED'
MQCA_CLUSTER_NAMELIST	2030	X'000007EE'
MQCA_CLUSTER_Q_MGR_NAME	2031	X'000007EF'
MQCA_CLUSTER_TIME	2038	X'000007F6'
MQCA_CLUSTER_WORKLOAD_DATA	2034	X'000007F2'
MQCA_CLUSTER_WORKLOAD_EXIT	2033	X'000007F1'
MQCA_COMMAND_INPUT_Q_NAME	2003	X'000007D3'
MQCA_COMMAND_REPLY_Q_NAME	2067	X'00000813'
MQCA_CREATION_DATE	2004	X'000007D4'
MQCA_CREATION_TIME	2005	X'000007D5'
MQCA_DEAD_LETTER_Q_NAME	2006	X'000007D6'
MQCA_DEF_XMIT_Q_NAME	2025	X'000007E9'
MQCA_DNS_GROUP	2071	X'00000817'
MQCA_ENV_DATA	2007	X'000007D7'
MQCA_FIRST	2001	X'000007D1'
MQCA_IGQ_USER_ID	2041	X'000007F9'

MQCA_INITIATION_Q_NAME	2008	X'000007D8'
MQCA_LAST	4000	X'00000FA0'
MQCA_LAST_USED	2109	X'0000083D'
MQCA_LDAP_PASSWORD	2048	X'00000800'
MQCA_LDAP_USER_NAME	2047	X'000007FF'
MQCA_LU_GROUP_NAME	2072	X'00000818'
MQCA_LU_NAME	2073	X'00000819'
MQCA_LU62_ARM_SUFFIX	2074	X'0000081A'
MQCA_MODEL_DURABLE_Q	2096	X'00000830'
MQCA_MODEL_NON_DURABLE_Q	2097	X'00000831'
MQCA_MONITOR_Q_NAME	2066	X'00000812'
MQCA_NAMELIST_DESC	2009	X'000007D9'
MQCA_NAMELIST_NAME	2010	X'000007DA'
MQCA_NAMES	2020	X'000007E4'
MQCA_PARENT	2102	X'00000836'
MQCA_PASS_TICKET_APPL	2086	X'00000826'
MQCA_PROCESS_DESC	2011	X'000007DB'
MQCA_PROCESS_NAME	2012	X'000007DC'
MQCA_Q_DESC	2013	X'000007DD'
MQCA_Q_MGR_DESC	2014	X'000007DE'
MQCA_Q_MGR_IDENTIFIER	2032	X'000007F0'
MQCA_Q_MGR_NAME	2015	X'000007DF'
MQCA_Q_NAME	2016	X'000007E0'
MQCA_QSG_NAME	2040	X'000007F8'
MQCA_REMOTE_Q_MGR_NAME	2017	X'000007E1'
MQCA_REMOTE_Q_NAME	2018	X'000007E2'
MQCA_REPOSITORY_NAME	2035	X'000007F3'
MQCA_REPOSITORY_NAMELIST	2036	X'000007F4'
MQCA_RESUME_DATE	2098	X'00000832'
MQCA_RESUME_TIME	2099	X'00000833'
MQCA_SERVICE_DESC	2078	X'0000081E'
MQCA_SERVICE_NAME	2077	X'0000081D'
MQCA_SERVICE_START_ARGS	2080	X'00000820'
MQCA_SERVICE_START_COMMAND	2079	X'0000081F'
MQCA_SERVICE_STOP_ARGS	2082	X'00000822'
MQCA_SERVICE_STOP_COMMAND	2081	X'00000821'
MQCA_STDERR_DESTINATION	2084	X'00000824'
MQCA_STDOUT_DESTINATION	2083	X'00000823'
MQCA_SSL_CRL_NAMELIST	2050	X'00000802'
MQCA_SSL_CRYPTO_HARDWARE	2051	X'00000803'
MQCA_SSL_KEY_LIBRARY	2069	X'00000815'
MQCA_SSL_KEY_MEMBER	2070	X'00000816'

MQCA_SSL_KEY_REPOSITORY	2049	X'00000801'
MQCA_STORAGE_CLASS	2022	X'000007E6'
MQCA_STORAGE_CLASS_DESC	2042	X'000007FA'
MQCA_SYSTEM_LOG_Q_NAME	2065	X'00000811'
MQCA_TCP_NAME	2075	X'0000081B'
MQCA_TOPIC_DESC	2093	X'0000082D'
MQCA_TOPIC_NAME	2092	X'0000082C'
MQCA_TOPIC_STRING_FILTER	2108	X'0000083C'
MQCA_TOPIC_STRING	2094	X'0000082E'
MQCA_TPIPE_NAME	2085	X'00000825'
MQCA_TRIGGER_CHANNEL_NAME	2064	X'00000810'
MQCA_TRIGGER_DATA	2023	X'000007E7'
MQCA_TRIGGER_PROGRAM_NAME	2062	X'0000080E'
MQCA_TRIGGER_TERM_ID	2063	X'0000080F'
MQCA_TRIGGER_TRANS_ID	2061	X'0000080D'
MQCA_USER_DATA	2021	X'000007E5'
MQCA_USER_LIST	4000	X'00000FA0'
MQCA_VERSION	2120	X'00000848'
MQCA_XCF_GROUP_NAME	2043	X'000007FB'
MQCA_XCF_MEMBER_NAME	2044	X'000007FC'
MQCA_XMIT_Q_NAME	2024	X'000007E8'

MQCACF_* (Command format Character Parameter Types):

MQCACF_FIRST	3001	X'00000BB9'
MQCACF_FROM_Q_NAME	3001	X'00000BB9'
MQCACF_TO_Q_NAME	3002	X'00000BBA'
MQCACF_FROM_PROCESS_NAME	3003	X'00000BBB'
MQCACF_TO_PROCESS_NAME	3004	X'00000BBC'
MQCACF_FROM_NAMELIST_NAME	3005	X'00000BBD'
MQCACF_TO_NAMELIST_NAME	3006	X'00000BBE'
MQCACF_FROM_CHANNEL_NAME	3007	X'00000BBF'
MQCACF_TO_CHANNEL_NAME	3008	X'00000BC0'
MQCACF_FROM_AUTH_INFO_NAME	3009	X'00000BC1'
MQCACF_TO_AUTH_INFO_NAME	3010	X'00000BC2'
MQCACF_Q_NAMES	3011	X'00000BC3'
MQCACF_PROCESS_NAMES	3012	X'00000BC4'
MQCACF_NAMELIST_NAMES	3013	X'00000BC5'
MQCACF_ESCAPE_TEXT	3014	X'00000BC6'
MQCACF_LOCAL_Q_NAMES	3015	X'00000BC7'
MQCACF_MODEL_Q_NAMES	3016	X'00000BC8'
MQCACF_ALIAS_Q_NAMES	3017	X'00000BC9'

MQCACF_REMOTE_Q_NAMES	3018	X'00000BCA'
MQCACF_SENDER_CHANNEL_NAMES	3019	X'00000BCB'
MQCACF_SERVER_CHANNEL_NAMES	3020	X'00000BCC'
MQCACF_REQUESTER_CHANNEL_NAMES	3021	X'00000BCD'
MQCACF_RECEIVER_CHANNEL_NAMES	3022	X'00000BCE'
MQCACF_OBJECT_Q_MGR_NAME	3023	X'00000BCF'
MQCACF_APPL_NAME	3024	X'00000BD0'
MQCACF_USER_IDENTIFIER	3025	X'00000BD1'
MQCACF_AUX_ERROR_DATA_STR_1	3026	X'00000BD2'
MQCACF_AUX_ERROR_DATA_STR_2	3027	X'00000BD3'
MQCACF_AUX_ERROR_DATA_STR_3	3028	X'00000BD4'
MQCACF_BRIDGE_NAME	3029	X'00000BD5'
MQCACF_STREAM_NAME	3030	X'00000BD6'
MQCACF_TOPIC	3031	X'00000BD7'
MQCACF_PARENT_Q_MGR_NAME	3032	X'00000BD8'
MQCACF_CORREL_ID	3033	X'00000BD9'
MQCACF_PUBLISH_TIMESTAMP	3034	X'00000BDA'
MQCACF_STRING_DATA	3035	X'00000BDB'
MQCACF_SUPPORTED_STREAM_NAME	3036	X'00000BDC'
MQCACF_REG_TOPIC	3037	X'00000BDD'
MQCACF_REG_TIME	3038	X'00000BDE'
MQCACF_REG_USER_ID	3039	X'00000BDF'
MQCACF_CHILD_Q_MGR_NAME	3040	X'00000BE0'
MQCACF_REG_STREAM_NAME	3041	X'00000BE1'
MQCACF_REG_Q_MGR_NAME	3042	X'00000BE2'
MQCACF_REG_Q_NAME	3043	X'00000BE3'
MQCACF_REG_CORREL_ID	3044	X'00000BE4'
MQCACF_EVENT_USER_ID	3045	X'00000BE5'
MQCACF_OBJECT_NAME	3046	X'00000BE6'
MQCACF_EVENT_Q_MGR	3047	X'00000BE7'
MQCACF_AUTH_INFO_NAMES	3048	X'00000BE8'
MQCACF_EVENT_APPL_IDENTITY	3049	X'00000BE9'
MQCACF_EVENT_APPL_NAME	3050	X'00000BEA'
MQCACF_EVENT_APPL_ORIGIN	3051	X'00000BEB'
MQCACF_SUBSCRIPTION_NAME	3052	X'00000BEC'
MQCACF_REG_SUB_NAME	3053	X'00000BED'
MQCACF_SUBSCRIPTION_IDENTITY	3054	X'00000BEE'
MQCACF_REG_SUB_IDENTITY	3055	X'00000BEF'
MQCACF_SUBSCRIPTION_USER_DATA	3056	X'00000BF0'
MQCACF_REG_SUB_USER_DATA	3057	X'00000BF1'
MQCACF_APPL_TAG	3058	X'00000BF2'
MQCACF_DATA_SET_NAME	3059	X'00000BF3'

MQCACF_UOW_START_DATE	3060	X'00000BF4'
MQCACF_UOW_START_TIME	3061	X'00000BF5'
MQCACF_UOW_LOG_START_DATE	3062	X'00000BF6'
MQCACF_UOW_LOG_START_TIME	3063	X'00000BF7'
MQCACF_UOW_LOG_EXTENT_NAME	3064	X'00000BF8'
MQCACF_PRINCIPAL_ENTITY_NAMES	3065	X'00000BF9'
MQCACF_GROUP_ENTITY_NAMES	3066	X'00000BFA'
MQCACF_AUTH_PROFILE_NAME	3067	X'00000BFB'
MQCACF_ENTITY_NAME	3068	X'00000BFC'
MQCACF_SERVICE_COMPONENT	3069	X'00000BFD'
MQCACF_RESPONSE_Q_MGR_NAME	3070	X'00000BFE'
MQCACF_CURRENT_LOG_EXTENT_NAME	3071	X'00000BFF'
MQCACF_RESTART_LOG_EXTENT_NAME	3072	X'00000C00'
MQCACF_MEDIA_LOG_EXTENT_NAME	3073	X'00000C01'
MQCACF_LOG_PATH	3074	X'00000C02'
MQCACF_COMMAND_MQSC	3075	X'00000C03'
MQCACF_Q_MGR_CPF	3076	X'00000C04'
MQCACF_USAGE_LOG_RBA	3078	X'00000C06'
MQCACF_USAGE_LOG_LRSN	3079	X'00000C07'
MQCACF_COMMAND_SCOPE	3080	X'00000C08'
MQCACF_ASID	3081	X'00000C09'
MQCACF_PSB_NAME	3082	X'00000C0A'
MQCACF_PST_ID	3083	X'00000C0B'
MQCACF_TASK_NUMBER	3084	X'00000C0C'
MQCACF_TRANSACTION_ID	3085	X'00000C0D'
MQCACF_Q_MGR_UOW_ID	3086	X'00000C0E'
MQCACF_ORIGIN_NAME	3088	X'00000C10'
MQCACF_ENV_INFO	3089	X'00000C11'
MQCACF_SECURITY_PROFILE	3090	X'00000C12'
MQCACF_CONFIGURATION_DATE	3091	X'00000C13'
MQCACF_CONFIGURATION_TIME	3092	X'00000C14'
MQCACF_FROM_CF_STRUC_NAME	3093	X'00000C15'
MQCACF_TO_CF_STRUC_NAME	3094	X'00000C16'
MQCACF_CF_STRUC_NAMES	3095	X'00000C17'
MQCACF_FAIL_DATE	3096	X'00000C18'
MQCACF_FAIL_TIME	3097	X'00000C19'
MQCACF_BACKUP_DATE	3098	X'00000C1A'
MQCACF_BACKUP_TIME	3099	X'00000C1B'
MQCACF_SYSTEM_NAME	3100	X'00000C1C'
MQCACF_CF_STRUC_BACKUP_START	3101	X'00000C1D'
MQCACF_CF_STRUC_BACKUP_END	3102	X'00000C1E'
MQCACF_CF_STRUC_LOG_Q_MGRS	3103	X'00000C1F'

MQCACF_FROM_STORAGE_CLASS	3104	X'00000C20'
MQCACF_TO_STORAGE_CLASS	3105	X'00000C21'
MQCACF_STORAGE_CLASS_NAMES	3106	X'00000C22'
MQCACF_DSG_NAME	3108	X'00000C24'
MQCACF_DB2_NAME	3109	X'00000C25'
MQCACF_SYSP_CMD_USER_ID	3110	X'00000C26'
MQCACF_SYSP_OTMA_GROUP	3111	X'00000C27'
MQCACF_SYSP_OTMA_MEMBER	3112	X'00000C28'
MQCACF_SYSP_OTMA_DRU_EXIT	3113	X'00000C29'
MQCACF_SYSP_OTMA_TPIPE_PFX	3114	X'00000C2A'
MQCACF_SYSP_ARCHIVE_PFX1	3115	X'00000C2B'
MQCACF_SYSP_ARCHIVE_UNIT1	3116	X'00000C2C'
MQCACF_SYSP_LOG_CORREL_ID	3117	X'00000C2D'
MQCACF_SYSP_UNIT_VOLSER	3118	X'00000C2E'
MQCACF_SYSP_Q_MGR_TIME	3119	X'00000C2F'
MQCACF_SYSP_Q_MGR_DATE	3120	X'00000C30'
MQCACF_SYSP_Q_MGR_RBA	3121	X'00000C31'
MQCACF_SYSP_LOG_RBA	3122	X'00000C32'
MQCACF_SYSP_SERVICE	3123	X'00000C33'
MQCACF_FROM_LISTENER_NAME	3124	X'00000C34'
MQCACF_TO_LISTENER_NAME	3125	X'00000C35'
MQCACF_FROM_SERVICE_NAME	3126	X'00000C36'
MQCACF_TO_SERVICE_NAME	3127	X'00000C37'
MQCACF_LAST_PUT_DATE	3128	X'00000C38'
MQCACF_LAST_PUT_TIME	3129	X'00000C39'
MQCACF_LAST_GET_DATE	3130	X'00000C3A'
MQCACF_LAST_GET_TIME	3131	X'00000C3B'
MQCACF_OPERATION_DATE	3132	X'00000C3C'
MQCACF_OPERATION_TIME	3133	X'00000C3D'
MQCACF_ACTIVITY_DESC	3134	X'00000C3E'
MQCACF_APPL_IDENTITY_DATA	3135	X'00000C3F'
MQCACF_APPL_ORIGIN_DATA	3136	X'00000C40'
MQCACF_PUT_DATE	3137	X'00000C41'
MQCACF_PUT_TIME	3138	X'00000C42'
MQCACF_REPLY_TO_Q	3139	X'00000C43'
MQCACF_REPLY_TO_Q_MGR	3140	X'00000C44'
MQCACF_RESOLVED_Q_NAME	3141	X'00000C45'
MQCACF_STRUC_ID	3142	X'00000C46'
MQCACF_VALUE_NAME	3143	X'00000C47'
MQCACF_SERVICE_START_DATE	3144	X'00000C48'
MQCACF_SERVICE_START_TIME	3145	X'00000C49'
MQCACF_SYSP_OFFLINE_RBA	3146	X'00000C4A'

MQCACF_SYSP_ARCHIVE_PFX2	3147	X'00000C4B'
MQCACF_SYSP_ARCHIVE_UNIT2	3148	X'00000C4C'
MQCACF_TO_TOPIC_NAME	3149	X'00000C4D'
MQCACF_FROM_TOPIC_NAME	3150	X'00000C4E'
MQCACF_TOPIC_NAMES	3151	X'00000C4F'
MQCACF_SUB_NAME	3152	X'00000C50'
MQCACF_DESTINATION_Q_MGR	3153	X'00000C51'
MQCACF_DESTINATION	3154	X'00000C52'
MQCACF_SUB_USER_ID	3156	X'00000C54'
MQCACF_SUB_USER_DATA	3159	X'00000C57'
MQCACF_SUB_SELECTOR	3160	X'00000C58'
MQCACF_LAST_PUB_DATE	3161	X'00000C59'
MQCACF_LAST_PUB_TIME	3162	X'00000C5A'
MQCACF_FROM_SUB_NAME	3163	X'00000C5B'
MQCACF_TO_SUB_NAME	3164	X'00000C5C'
MQCACF_LAST_MSG_TIME	3167	X'00000C5F'
MQCACF_LAST_MSG_DATE	3168	X'00000C60'
MQCACF_SUBSCRIPTION_POINT	3169	X'00000C61'
MQCACF_FILTER	3170	X'00000C62'
MQCACF_NONE	3171	X'00000C63'
MQCACF_ADMIN_TOPIC_NAMES	3172	X'00000C64'
MQCACF_LAST_USED	3172	X'00000C64'

MQCACH_* (Command format Character Channel Parameter Types):

MQCACH_FIRST	3501	X'00000DAD'
MQCACH_CHANNEL_NAME	3501	X'00000DAD'
MQCACH_DESC	3502	X'00000DAE'
MQCACH_MODE_NAME	3503	X'00000DAF'
MQCACH_TP_NAME	3504	X'00000DB0'
MQCACH_XMIT_Q_NAME	3505	X'00000DB1'
MQCACH_CONNECTION_NAME	3506	X'00000DB2'
MQCACH_MCA_NAME	3507	X'00000DB3'
MQCACH_SEC_EXIT_NAME	3508	X'00000DB4'
MQCACH_MSG_EXIT_NAME	3509	X'00000DB5'
MQCACH_SEND_EXIT_NAME	3510	X'00000DB6'
MQCACH_RCV_EXIT_NAME	3511	X'00000DB7'
MQCACH_CHANNEL_NAMES	3512	X'00000DB8'
MQCACH_SEC_EXIT_USER_DATA	3513	X'00000DB9'
MQCACH_MSG_EXIT_USER_DATA	3514	X'00000DBA'
MQCACH_SEND_EXIT_USER_DATA	3515	X'00000DBB'
MQCACH_RCV_EXIT_USER_DATA	3516	X'00000DBC'

MQCACH_USER_ID	3517	X'00000DBD'
MQCACH_PASSWORD	3518	X'00000DBE'
MQCACH_LOCAL_ADDRESS	3520	X'00000DC0'
MQCACH_LOCAL_NAME	3521	X'00000DC1'
MQCACH_LAST_MSG_TIME	3524	X'00000DC4'
MQCACH_LAST_MSG_DATE	3525	X'00000DC5'
MQCACH_MCA_USER_ID	3527	X'00000DC7'
MQCACH_CHANNEL_START_TIME	3528	X'00000DC8'
MQCACH_CHANNEL_START_DATE	3529	X'00000DC9'
MQCACH_MCA_JOB_NAME	3530	X'00000DCA'
MQCACH_LAST_LUWID	3531	X'00000DCB'
MQCACH_CURRENT_LUWID	3532	X'00000DCC'
MQCACH_FORMAT_NAME	3533	X'00000DCD'
MQCACH_MR_EXIT_NAME	3534	X'00000DCE'
MQCACH_MR_EXIT_USER_DATA	3535	X'00000DCF'
MQCACH_SSL_CIPHER_SPEC	3544	X'00000DD8'
MQCACH_SSL_PEER_NAME	3545	X'00000DD9'
MQCACH_SSL_HANDSHAKE_STAGE	3546	X'00000DDA'
MQCACH_SSL_SHORT_PEER_NAME	3547	X'00000DDB'
MQCACH_REMOTE_APPL_TAG	3548	X'00000DDC'
MQCACH_SSL_CERT_USER_ID	3549	X'00000DDD'
MQCACH_SSL_CERT_ISSUER_NAME	3550	X'00000DDE'
MQCACH_LU_NAME	3551	X'00000DDF'
MQCACH_IP_ADDRESS	3552	X'00000DE0'
MQCACH_TCP_NAME	3553	X'00000DE1'
MQCACH_LISTENER_NAME	3554	X'00000DE2'
MQCACH_LISTENER_DESC	3555	X'00000DE3'
MQCACH_LISTENER_START_DATE	3556	X'00000DE4'
MQCACH_LISTENER_START_TIME	3557	X'00000DE5'
MQCACH_SSL_KEY_RESET_DATE	3558	X'00000DE6'
MQCACH_SSL_KEY_RESET_TIME	3559	X'00000DE7'
MQCACH_LAST_USED	3559	X'00000DE7'

MQCADSD_* (CICS information header ADS Descriptors):

MQCADSD_NONE	0	X'00000000'
MQCADSD_SEND	1	X'00000001'
MQCADSD_RECV	16	X'00000010'
MQCADSD_MSGFORMAT	256	X'00000100'

MQCAFTY_* (Connection Affinity Values):

MQCAFTY_NONE	0	X'00000000'
MQCAFTY_PREFERRED	1	X'00000001'

MQCAMO_* (Command format Character Monitoring Parameter Types):

MQCAMO_FIRST	2701	X'00000A8D'
MQCAMO_CLOSE_DATE	2701	X'00000A8D'
MQCAMO_CLOSE_TIME	2702	X'00000A8E'
MQCAMO_CONN_DATE	2703	X'00000A8F'
MQCAMO_CONN_TIME	2704	X'00000A90'
MQCAMO_DISC_DATE	2705	X'00000A91'
MQCAMO_DISC_TIME	2706	X'00000A92'
MQCAMO_END_DATE	2707	X'00000A93'
MQCAMO_END_TIME	2708	X'00000A94'
MQCAMO_OPEN_DATE	2709	X'00000A95'
MQCAMO_OPEN_TIME	2710	X'00000A96'
MQCAMO_START_DATE	2711	X'00000A97'
MQCAMO_START_TIME	2712	X'00000A98'
MQCAMO_LAST_USED	2712	X'00000A98'

MQCBC_* (MQCBC constants structure):

MQCBC_STRUC_ID	"CBCb"	
MQCBC_STRUC_ID_ARRAY	'C','B','C','b'	
MQCBC_VERSION_1	1	X'00000001'
MQCBC_CURRENT_VERSION	1	X'00000001'

MQCBCF_* (MQCBC constants Flags):

MQCBCF_NONE	0	X'00000000'
MQCBCF_READA_BUFFER_EMPTY	1	X'00000001'

MQCBCT_* (MQCBC constants Callback type):

MQCBCT_START_CALL	1	X'00000001'
MQCBCT_STOP_CALL	2	X'00000002'
MQCBCT_REGISTER_CALL	3	X'00000003'
MQCBCT_DEREGISTER_CALL	4	X'00000004'
MQCBCT_EVENT_CALL	5	X'00000005'
MQCBCT_MSG_REMOVED	6	X'00000006'
MQCBCT_MSG_NOT_REMOVED	7	X'00000007'

MQCBD_* (MQCBD constants structure):

MQCBD_STRUC_ID	"CBDb"	
MQCBD_STRUC_ID_ARRAY	'C','B','D','b'	
MQCBD_VERSION_1	1	X'00000001'
MQCBD_CURRENT_VERSION	1	X'00000001'

MQCBDO_* (MQCBD constants Callback Options):

MQCBDO_NONE	0	X'00000000'
MQCBDO_START_CALL	1	X'00000001'
MQCBDO_STOP_CALL	4	X'00000004'
MQCBDO_REGISTER_CALL	256	X'00000100'
MQCBDO_DEREGISTER_CALL	512	X'00000200'
MQCBDO_FAIL_IF_QUIESCING	8192	X'00002000'

MQCBO_* (Create-Bag Options for mqCreateBag):

MQCBO_NONE	0	X'00000000'
MQCBO_USER_BAG	0	X'00000000'
MQCBO_ADMIN_BAG	1	X'00000001'
MQCBO_COMMAND_BAG	16	X'00000010'
MQCBO_SYSTEM_BAG	32	X'00000020'
MQCBO_GROUP_BAG	64	X'00000040'
MQCBO_LIST_FORM_ALLOWED	2	X'00000002'
MQCBO_LIST_FORM_INHIBITED	0	X'00000000'
MQCBO_REORDER_AS_REQUIRED	4	X'00000004'
MQCBO_DO_NOT_REORDER	0	X'00000000'
MQCBO_CHECK_SELECTORS	8	X'00000008'
MQCBO_DO_NOT_CHECK_SELECTORS	0	X'00000000'

MQCBT_* (MQCBD constants This is the type of the Callback Function):

MQCBT_MESSAGE_CONSUMER	1	X'00000001'
MQCBT_EVENT_HANDLER	2	X'00000002'

MQCC_* (Completion Codes):

MQCC_OK	0	X'00000000'
MQCC_WARNING	1	X'00000001'
MQCC_FAILED	2	X'00000002'
MQCC_UNKNOWN	-1	X'FFFFFFFF'

MQCCSI_* (Coded Character Set Identifiers):

MQCCSI_UNDEFINED	0	X'00000000'
MQCCSI_DEFAULT	0	X'00000000'
MQCCSI_Q_MGR	0	X'00000000'
MQCCSI_INHERIT	-2	X'FFFFFFFE'
MQCCSI_EMBEDDED	-1	X'FFFFFFFF'
MQCCSI_APPL	-3	X'FFFFFFFD'

MQCCT_* (CICS information header Conversational Task Options):

MQCCT_YES	1	X'00000001'
MQCCT_NO	0	X'00000000'

MQCD_* (Channel definition structure):

MQCD_VERSION_1	1	X'00000001'
MQCD_VERSION_2	2	X'00000002'
MQCD_VERSION_3	3	X'00000003'
MQCD_VERSION_4	4	X'00000004'
MQCD_VERSION_5	5	X'00000005'
MQCD_VERSION_6	6	X'00000006'
MQCD_VERSION_7	7	X'00000007'
MQCD_VERSION_8	8	X'00000008'
MQCD_VERSION_9	9	X'00000009'
MQCD_CURRENT_VERSION	9	X'00000009'
MQCD_LENGTH_4	(value differs by platform or version)	
MQCD_LENGTH_5	(value differs by platform or version)	
MQCD_LENGTH_6	(value differs by platform or version)	
MQCD_LENGTH_7	(value differs by platform or version)	
MQCD_LENGTH_8	(value differs by platform or version)	
MQCD_LENGTH_9	(value differs by platform or version)	
MQCD_CURRENT_LENGTH	(value differs by platform or version)	

MQCDC_* (Channel Data Conversion):

MQCDC_SENDER_CONVERSION	1	X'00000001'
MQCDC_NO_SENDER_CONVERSION	0	X'00000000'

MQCERT_* (Certificate Validation Policy Type):

MQ_CERT_VAL_POLICY_DEFAULT	0	X'00000000'
MQ_CERT_VAL_POLICY_ANY	0	X'00000000'
MQ_CERT_VAL_POLICY_RFC5280	1	X'00000001'

MQCF_* (Capability Flags):

MQCF_NONE	0	X'00000000'
MQCF_DIST_LISTS	1	X'00000001'

MQCFAC_* (CICS information header Facility):

MQCFAC_NONE	X'00...00'	(8 nulls)
MQCFAC_NONE_ARRAY	'\0','\0',...	(8 nulls)

MQCFBF_* (Command format byte string filter parameter structure):

MQCFBF_STRUC_LENGTH_FIXED	20	X'00000014'

MQCFBS_* (Command format byte string parameter structure):

MQCFBS_STRUC_LENGTH_FIXED	16	X'00000010'

MQCFC_* (Command format header Control Options):

MQCFC_LAST	1	X'00000001'
MQCFC_NOT_LAST	0	X'00000000'

MQCFGR_* (Command format group parameter structure):

MQCFGR_STRUC_LENGTH	16	X'00000010'

MQCFH_* (Command format header structure):

MQCFH_STRUC_LENGTH	36	X'00000024'
MQCFH_VERSION_1	1	X'00000001'
MQCFH_VERSION_2	2	X'00000002'
MQCFH_VERSION_3	3	X'00000003'
MQCFH_CURRENT_VERSION	3	X'00000003'

MQCFIF_* (Command format integer filter parameter structure):

MQCFIF_STRUC_LENGTH	20	X'00000014'

MQCFIL_* (Command format integer list parameter structure):

MQCFIL_STRUC_LENGTH_FIXED	16	X'00000010'

MQCFIL64_* (Command format 64-bit integer list parameter structure):

MQCFIL64_STRUC_LENGTH_FIXED	16	X'00000010'

MQCFIN_* (Command format integer parameter structure):

MQCFIN_STRUC_LENGTH	16	X'00000010'

MQCFIN64_* (Command format 64-bit integer parameter structure):

MQCFIN64_STRUC_LENGTH	24	X'00000018'

MQCFO_* (Command format Refresh Repository Options and Command format Remove Queues Options):
Command format Refresh Repository Options

MQCFO_REFRESH_REPOSITORY_YES	1	X'00000001'
MQCFO_REFRESH_REPOSITORY_NO	0	X'00000000'

Command format Remove Queues Options

MQCFO_REMOVE_QUEUES_YES	1	X'00000001'
MQCFO_REMOVE_QUEUES_NO	0	X'00000000'

MQCFOP_* (Command format Filter Operators):

MQCFOP_LESS	1	X'00000001'
MQCFOP_EQUAL	2	X'00000002'
MQCFOP_GREATER	4	X'00000004'
MQCFOP_NOT_LESS	6	X'00000006'
MQCFOP_NOT_EQUAL	5	X'00000005'
MQCFOP_NOT_GREATER	3	X'00000003'
MQCFOP_LIKE	18	X'00000012'
MQCFOP_NOT_LIKE	21	X'00000015'
MQCFOP_CONTAINS	10	X'0000000A'
MQCFOP_EXCLUDES	13	X'0000000D'
MQCFOP_CONTAINS_GEN	26	X'0000001A'
MQCFOP_EXCLUDES_GEN	29	X'0000001D'

MQCFR_* (CF Recoverability):

MQCFR_YES	1	X'00000001'
MQCFR_NO	0	X'00000000'

MQCFSF_* (Command format string filter parameter structure):

MQCFSF_STRUC_LENGTH_FIXED	24	X'00000018'

MQCFSL_* (Command format string list parameter structure):

MQCFSL_STRUC_LENGTH_FIXED	24	X'00000018'

MQCFST_* (Command format string parameter structure):

MQCFST_STRUC_LENGTH_FIXED	20	X'00000014'

MQCFSTATUS_* (Command format CF Status):

MQCFSTATUS_NOT_FOUND	0	X'00000000'
MQCFSTATUS_ACTIVE	1	X'00000001'
MQCFSTATUS_IN_RECOVER	2	X'00000002'
MQCFSTATUS_IN_BACKUP	3	X'00000003'
MQCFSTATUS_FAILED	4	X'00000004'
MQCFSTATUS_NONE	5	X'00000005'
MQCFSTATUS_UNKNOWN	6	X'00000006'
MQCFSTATUS_ADMIN_INCOMPLETE	20	X'00000014'
MQCFSTATUS_NEVER_USED	21	X'00000015'
MQCFSTATUS_NO_BACKUP	22	X'00000016'
MQCFSTATUS_NOT_FAILED	23	X'00000017'
MQCFSTATUS_NOT_RECOVERABLE	24	X'00000018'
MQCFSTATUS_XES_ERROR	25	X'00000019'

MQCFT_* (Command format Types of Structure):

MQCFT_NONE	0	X'00000000'
MQCFT_COMMAND	1	X'00000001'
MQCFT_RESPONSE	2	X'00000002'
MQCFT_INTEGER	3	X'00000003'
MQCFT_STRING	4	X'00000004'
MQCFT_INTEGER_LIST	5	X'00000005'
MQCFT_STRING_LIST	6	X'00000006'
MQCFT_EVENT	7	X'00000007'
MQCFT_USER	8	X'00000008'
MQCFT_BYTE_STRING	9	X'00000009'
MQCFT_TRACE_ROUTE	10	X'0000000A'

MQCFT_REPORT	12	X'0000000C'
MQCFT_INTEGER_FILTER	13	X'0000000D'
MQCFT_STRING_FILTER	14	X'0000000E'
MQCFT_BYTE_STRING_FILTER	15	X'0000000F'
MQCFT_COMMAND_XR	16	X'00000010'
MQCFT_XR_MSG	17	X'00000011'
MQCFT_XR_ITEM	18	X'00000012'
MQCFT_XR_SUMMARY	19	X'00000013'
MQCFT_GROUP	20	X'00000014'
MQCFT_STATISTICS	21	X'00000015'
MQCFT_ACCOUNTING	22	X'00000016'
MQCFT_INTEGER64	23	X'00000017'
MQCFT_INTEGER64_LIST	25	X'00000019'

MQCFTYPE_* (Command format CF Types):

MQCFTYPE_APPL	0	X'00000000'
MQCFTYPE_ADMIN	1	X'00000001'

MQCFUNC_* (CICS information header Functions):

MQCFUNC_MQCONN	"CONN"	
MQCFUNC_MQGET	"GETb"	
MQCFUNC_MQINQ	"INQb"	
MQCFUNC_MQOPEN	"OPEN"	
MQCFUNC_MQPUT	"PUTb"	
MQCFUNC_MQPUT1	"PUT1"	
MQCFUNC_NONE	"bbbb"	
MQCFUNC_MQCONN_ARRAY	'C','O','N','N'	
MQCFUNC_MQGET_ARRAY	'G','E','T','b'	
MQCFUNC_MQINQ_ARRAY	'I','N','Q','b'	
MQCFUNC_MQOPEN_ARRAY	'O','P','E','N'	
MQCFUNC_MQPUT_ARRAY	'P','U','T','b'	
MQCFUNC_MQPUT1_ARRAY	'P','U','T','1'	
MQCFUNC_NONE_ARRAY	'b','b','b','b'	

MQCGWI_* (CICS information header Get Wait Interval):

MQCGWI_DEFAULT	-2	X'FFFFFFFE'

MQCHAD_* (Channel Auto Definition):

MQCHAD_DISABLED	0	X'00000000'
MQCHAD_ENABLED	1	X'00000001'

MQCHIDS_* (Command format Indoubt Status):

MQCHIDS_NOT_INDOUBT	0	X'00000000'
MQCHIDS_INDOUBT	1	X'00000001'

MQCHLD_* (Command format Channel Dispositions):

MQCHLD_ALL	-1	X'FFFFFFFF'
MQCHLD_DEFAULT	1	X'00000001'
MQCHLD_SHARED	2	X'00000002'
MQCHLD_PRIVATE	4	X'00000004'
MQCHLD_FIXSHARED	5	X'00000005'

MQCHS_* (Command format Channel Status):

MQCHS_INACTIVE	0	X'00000000'
MQCHS_BINDING	1	X'00000001'
MQCHS_STARTING	2	X'00000002'
MQCHS_RUNNING	3	X'00000003'
MQCHS_STOPPING	4	X'00000004'
MQCHS_RETRYING	5	X'00000005'
MQCHS_STOPPED	6	X'00000006'
MQCHS_REQUESTING	7	X'00000007'
MQCHS_PAUSED	8	X'00000008'
MQCHS_INITIALIZING	13	X'0000000D'
MQCHS_SWITCHING	14	X'0000000E'

MQCHSH_* (Command format Channel Shared Restart Options):

MQCHSH_RESTART_NO	0	X'00000000'
MQCHSH_RESTART_YES	1	X'00000001'

MQCHSR_* (Command format Channel Stop Options):

MQCHSR_STOP_NOT_REQUESTED	0	X'00000000'
MQCHSR_STOP_REQUESTED	1	X'00000001'

MQCHSSTATE_* (Command format Channel Substates):

MQCHSSTATE_OTHER	0	X'00000000'
MQCHSSTATE_END_OF_BATCH	100	X'00000064'
MQCHSSTATE_SENDING	200	X'000000C8'
MQCHSSTATE_RECEIVING	300	X'0000012C'
MQCHSSTATE_SERIALIZING	400	X'00000190'
MQCHSSTATE_RESYNCHING	500	X'000001F4'
MQCHSSTATE_HEARTBEATING	600	X'00000258'
MQCHSSTATE_IN_SCYEXIT	700	X'000002BC'
MQCHSSTATE_IN_RCVEXIT	800	X'00000320'
MQCHSSTATE_IN_SENDEXIT	900	X'00000384'
MQCHSSTATE_IN_MSGEXIT	1000	X'000003E8'
MQCHSSTATE_IN_MREXIT	1100	X'0000044C'
MQCHSSTATE_IN_CHADEXIT	1200	X'000004B0'
MQCHSSTATE_NET_CONNECTING	1250	X'000004E2'
MQCHSSTATE_SSL_HANDSHAKING	1300	X'00000514'
MQCHSSTATE_NAME_SERVER	1400	X'00000578'
MQCHSSTATE_IN_MQPUT	1500	X'000005DC'
MQCHSSTATE_IN_MQGET	1600	X'00000640'
MQCHSSTATE_IN_MQI_CALL	1700	X'000006A4'
MQCHSSTATE_COMPRESSING	1800	X'00000708'

MQCHT_* (Channel Types):

MQCHT_SENDER	1	X'00000001'
MQCHT_SERVER	2	X'00000002'
MQCHT_RECEIVER	3	X'00000003'
MQCHT_REQUESTER	4	X'00000004'
MQCHT_ALL	5	X'00000005'
MQCHT_CLNTCONN	6	X'00000006'
MQCHT_SVRCONN	7	X'00000007'
MQCHT_CLUSRCVR	8	X'00000008'
MQCHT_CLUSSDR	9	X'00000009'

MQCHTAB_* (Command format Channel Table Types):

MQCHTAB_Q_MGR	1	X'00000001'
MQCHTAB_CLNTCONN	2	X'00000002'

MQCI_* (Correlation Identifier):

MQCI_NONE	X'00...00'	(24 nulls)
MQCI_NONE_ARRAY	'\0','\0',...	(24 nulls)
MQCI_NEW_SESSION	X'414D5121...'	
MQCI_NEW_SESSION_ARRAY	'\x41','\x4D','\51','\ x21',...	

MQCIH_* (CICS information header structure and Flags):
CICS information header structure

MQCIH_STRUC_ID	"CIHb"	
MQCIH_STRUC_ID_ARRAY	'C','I','H','b'	
MQCIH_VERSION_1	1	X'00000001'
MQCIH_VERSION_2	2	X'00000002'
MQCIH_CURRENT_VERSION	2	X'00000002'
MQCIH_LENGTH_1	164	X'000000A4'
MQCIH_LENGTH_2	180	X'000000B4'
MQCIH_CURRENT_LENGTH	180	X'000000B4'

CICS information header Flags

MQCIH_NONE	0	X'00000000'
MQCIH_PASS_EXPIRATION	1	X'00000001'
MQCIH_UNLIMITED_EXPIRATION	0	X'00000000'
MQCIH_REPLY_WITHOUT_NULLS	2	X'00000002'
MQCIH_REPLY_WITH_NULLS	0	X'00000000'
MQCIH_SYNC_ON_RETURN	4	X'00000004'
MQCIH_NO_SYNC_ON_RETURN	0	X'00000000'

MQCLCT_* (Cluster Cache Types):

MQCLCT_STATIC	0	X'00000000'
MQCLCT_DYNAMIC	1	X'00000001'

MQCLRS_* (Command format Clear Topic String Scope):

MQCLRS_LOCAL	1	X'00000001'
MQCLRS_GLOBAL	2	X'00000002'

MQCLRT_* (Command format Clear Topic String Type):

MQCLRT_RETAINED	1	X'00000001'

MQCLT_* (CICS information header Link Types):

MQCLT_PROGRAM	1	X'00000001'
MQCLT_TRANSACTION	2	X'00000002'

MQCLWL_* (Cluster Workload):

MQCLWL_USEQ_LOCAL	0	X'00000000'
MQCLWL_USEQ_ANY	1	X'00000001'
MQCLWL_USEQ_AS_Q_MGR	-3	X'FFFFFFFD'

MQCLXQ_* (Cluster transmission queue type):

MQCLXQ_* are the values you can set in the DEFCLXQ queue manager attribute. The DEFCLXQ attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

MQCLXQ_SCTQ	0	X'00000000'
MQCLXQ_CHANNEL	1	X'00000001'

Related reference:

"DefClusterXmitQueueType (MQLONG)" on page 2113
The DefClusterXmitQueueType attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

"MQINQ - Inquire object attributes" on page 2005
The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

Related information:

Change Queue Manager
The Change Queue Manager (MQCMD_CHANGE_Q_MGR) command changes the specified attributes of the queue manager.

Inquire Queue Manager
The Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command inquires about the attributes of a queue manager.

Inquire Queue Manager (Response)
The response to the Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) command consists of the response header followed by the *QMgrName* structure and the requested combination of attribute parameter structures.

MQCMD_* (Command Codes):

MQCMD_NONE	0	X'00000000'
MQCMD_CHANGE_Q_MGR	1	X'00000001'
MQCMD_INQUIRE_Q_MGR	2	X'00000002'
MQCMD_CHANGE_PROCESS	3	X'00000003'
MQCMD_COPY_PROCESS	4	X'00000004'
MQCMD_CREATE_PROCESS	5	X'00000005'
MQCMD_DELETE_PROCESS	6	X'00000006'
MQCMD_INQUIRE_PROCESS	7	X'00000007'
MQCMD_CHANGE_Q	8	X'00000008'
MQCMD_CLEAR_Q	9	X'00000009'
MQCMD_COPY_Q	10	X'0000000A'
MQCMD_CREATE_Q	11	X'0000000B'
MQCMD_DELETE_Q	12	X'0000000C'
MQCMD_INQUIRE_Q	13	X'0000000D'
MQCMD_REFRESH_Q_MGR	16	X'00000010'
MQCMD_RESET_Q_STATS	17	X'00000011'
MQCMD_INQUIRE_Q_NAMES	18	X'00000012'
MQCMD_INQUIRE_PROCESS_NAMES	19	X'00000013'
MQCMD_INQUIRE_CHANNEL_NAMES	20	X'00000014'
MQCMD_CHANGE_CHANNEL	21	X'00000015'
MQCMD_COPY_CHANNEL	22	X'00000016'
MQCMD_CREATE_CHANNEL	23	X'00000017'
MQCMD_DELETE_CHANNEL	24	X'00000018'
MQCMD_INQUIRE_CHANNEL	25	X'00000019'
MQCMD_PING_CHANNEL	26	X'0000001A'
MQCMD_RESET_CHANNEL	27	X'0000001B'
MQCMD_START_CHANNEL	28	X'0000001C'
MQCMD_STOP_CHANNEL	29	X'0000001D'
MQCMD_START_CHANNEL_INIT	30	X'0000001E'
MQCMD_START_CHANNEL_LISTENER	31	X'0000001F'
MQCMD_CHANGE_NAMELIST	32	X'00000020'
MQCMD_COPY_NAMELIST	33	X'00000021'
MQCMD_CREATE_NAMELIST	34	X'00000022'
MQCMD_DELETE_NAMELIST	35	X'00000023'
MQCMD_INQUIRE_NAMELIST	36	X'00000024'
MQCMD_INQUIRE_NAMELIST_NAMES	37	X'00000025'
MQCMD_ESCAPE	38	X'00000026'
MQCMD_RESOLVE_CHANNEL	39	X'00000027'
MQCMD_PING_Q_MGR	40	X'00000028'
MQCMD_INQUIRE_Q_STATUS	41	X'00000029'
MQCMD_INQUIRE_CHANNEL_STATUS	42	X'0000002A'
MQCMD_CONFIG_EVENT	43	X'0000002B'

MQCMD_Q_MGR_EVENT	44	X'0000002C'
MQCMD_PERFM_EVENT	45	X'0000002D'
MQCMD_CHANNEL_EVENT	46	X'0000002E'
MQCMD_DELETE_PUBLICATION	60	X'0000003C'
MQCMD_DEREGISTER_PUBLISHER	61	X'0000003D'
MQCMD_DEREGISTER_SUBSCRIBER	62	X'0000003E'
MQCMD_PUBLISH	63	X'0000003F'
MQCMD_REGISTER_PUBLISHER	64	X'00000040'
MQCMD_REGISTER_SUBSCRIBER	65	X'00000041'
MQCMD_REQUEST_UPDATE	66	X'00000042'
MQCMD_BROKER_INTERNAL	67	X'00000043'
MQCMD_ACTIVITY_MSG	69	X'00000045'
MQCMD_INQUIRE_CLUSTER_Q_MGR	70	X'00000046'
MQCMD_RESUME_Q_MGR_CLUSTER	71	X'00000047'
MQCMD_SUSPEND_Q_MGR_CLUSTER	72	X'00000048'
MQCMD_REFRESH_CLUSTER	73	X'00000049'
MQCMD_RESET_CLUSTER	74	X'0000004A'
MQCMD_TRACE_ROUTE	75	X'0000004B'
MQCMD_REFRESH_SECURITY	78	X'0000004E'
MQCMD_CHANGE_AUTH_INFO	79	X'0000004F'
MQCMD_COPY_AUTH_INFO	80	X'00000050'
MQCMD_CREATE_AUTH_INFO	81	X'00000051'
MQCMD_DELETE_AUTH_INFO	82	X'00000052'
MQCMD_INQUIRE_AUTH_INFO	83	X'00000053'
MQCMD_INQUIRE_AUTH_INFO_NAMES	84	X'00000054'
MQCMD_INQUIRE_CONNECTION	85	X'00000055'
MQCMD_STOP_CONNECTION	86	X'00000056'
MQCMD_INQUIRE_AUTH_RECS	87	X'00000057'
MQCMD_INQUIRE_ENTITY_AUTH	88	X'00000058'
MQCMD_DELETE_AUTH_REC	89	X'00000059'
MQCMD_SET_AUTH_REC	90	X'0000005A'
MQCMD_LOGGER_EVENT	91	X'0000005B'
MQCMD_RESET_Q_MGR	92	X'0000005C'
MQCMD_CHANGE_LISTENER	93	X'0000005D'
MQCMD_COPY_LISTENER	94	X'0000005E'
MQCMD_CREATE_LISTENER	95	X'0000005F'
MQCMD_DELETE_LISTENER	96	X'00000060'
MQCMD_INQUIRE_LISTENER	97	X'00000061'
MQCMD_INQUIRE_LISTENER_STATUS	98	X'00000062'
MQCMD_COMMAND_EVENT	99	X'00000063'
MQCMD_CHANGE_SECURITY	100	X'00000064'
MQCMD_CHANGE_CF_STRUC	101	X'00000065'

MQCMD_CHANGE_STG_CLASS	102	X'00000066'
MQCMD_CHANGE_TRACE	103	X'00000067'
MQCMD_ARCHIVE_LOG	104	X'00000068'
MQCMD_BACKUP_CF_STRUC	105	X'00000069'
MQCMD_CREATE_BUFFER_POOL	106	X'0000006A'
MQCMD_CREATE_PAGE_SET	107	X'0000006B'
MQCMD_CREATE_CF_STRUC	108	X'0000006C'
MQCMD_CREATE_STG_CLASS	109	X'0000006D'
MQCMD_COPY_CF_STRUC	110	X'0000006E'
MQCMD_COPY_STG_CLASS	111	X'0000006F'
MQCMD_DELETE_CF_STRUC	112	X'00000070'
MQCMD_DELETE_STG_CLASS	113	X'00000071'
MQCMD_INQUIRE_ARCHIVE	114	X'00000072'
MQCMD_INQUIRE_CF_STRUC	115	X'00000073'
MQCMD_INQUIRE_CF_STRUC_STATUS	116	X'00000074'
MQCMD_INQUIRE_CMD_SERVER	117	X'00000075'
MQCMD_INQUIRE_CHANNEL_INIT	118	X'00000076'
MQCMD_INQUIRE_QSG	119	X'00000077'
MQCMD_INQUIRE_LOG	120	X'00000078'
MQCMD_INQUIRE_SECURITY	121	X'00000079'
MQCMD_INQUIRE_STG_CLASS	122	X'0000007A'
MQCMD_INQUIRE_SYSTEM	123	X'0000007B'
MQCMD_INQUIRE_THREAD	124	X'0000007C'
MQCMD_INQUIRE_TRACE	125	X'0000007D'
MQCMD_INQUIRE_USAGE	126	X'0000007E'
MQCMD_MOVE_Q	127	X'0000007F'
MQCMD_RECOVER_BSDS	128	X'00000080'
MQCMD_RECOVER_CF_STRUC	129	X'00000081'
MQCMD_RESET_TPIPE	130	X'00000082'
MQCMD_RESOLVE_INDOUBT	131	X'00000083'
MQCMD_RESUME_Q_MGR	132	X'00000084'
MQCMD_REVERIFY_SECURITY	133	X'00000085'
MQCMD_SET_ARCHIVE	134	X'00000086'
MQCMD_SET_LOG	136	X'00000088'
MQCMD_SET_SYSTEM	137	X'00000089'
MQCMD_START_CMD_SERVER	138	X'0000008A'
MQCMD_START_Q_MGR	139	X'0000008B'
MQCMD_START_TRACE	140	X'0000008C'
MQCMD_STOP_CHANNEL_INIT	141	X'0000008D'
MQCMD_STOP_CHANNEL_LISTENER	142	X'0000008E'
MQCMD_STOP_CMD_SERVER	143	X'0000008F'
MQCMD_STOP_Q_MGR	144	X'00000090'

MQCMD_STOP_TRACE	145	X'00000091'
MQCMD_SUSPEND_Q_MGR	146	X'00000092'
MQCMD_INQUIRE_CF_STRUC_NAMES	147	X'00000093'
MQCMD_INQUIRE_STG_CLASS_NAMES	148	X'00000094'
MQCMD_CHANGE_SERVICE	149	X'00000095'
MQCMD_COPY_SERVICE	150	X'00000096'
MQCMD_CREATE_SERVICE	151	X'00000097'
MQCMD_DELETE_SERVICE	152	X'00000098'
MQCMD_INQUIRE_SERVICE	153	X'00000099'
MQCMD_INQUIRE_SERVICE_STATUS	154	X'0000009A'
MQCMD_START_SERVICE	155	X'0000009B'
MQCMD_STOP_SERVICE	156	X'0000009C'
MQCMD_DELETE_BUFFER_POOL	157	X'0000009D'
MQCMD_DELETE_PAGE_SET	158	X'0000009E'
MQCMD_CHANGE_BUFFER_POOL	159	X'0000009F'
MQCMD_CHANGE_PAGE_SET	160	X'000000A0'
MQCMD_INQUIRE_Q_MGR_STATUS	161	X'000000A1'
MQCMD_CREATE_LOG	162	X'000000A2'
MQCMD_STATISTICS_MQI	164	X'000000A4'
MQCMD_STATISTICS_Q	165	X'000000A5'
MQCMD_STATISTICS_CHANNEL	166	X'000000A6'
MQCMD_ACCOUNTING_MQI	167	X'000000A7'
MQCMD_ACCOUNTING_Q	168	X'000000A8'
MQCMD_INQUIRE_AUTH_SERVICE	169	X'000000A9'
MQCMD_CHANGE_TOPIC	170	X'000000AA'
MQCMD_COPY_TOPIC	171	X'000000AB'
MQCMD_CREATE_TOPIC	172	X'000000AC'
MQCMD_DELETE_TOPIC	173	X'000000AD'
MQCMD_INQUIRE_TOPIC	174	X'000000AE'
MQCMD_INQUIRE_TOPIC_NAMES	175	X'000000AF'
MQCMD_INQUIRE_SUBSCRIPTION	176	X'000000B0'
MQCMD_CREATE_SUBSCRIPTION	177	X'000000B1'
MQCMD_CHANGE_SUBSCRIPTION	178	X'000000B2'
MQCMD_DELETE_SUBSCRIPTION	179	X'000000B3'
MQCMD_COPY_SUBSCRIPTION	181	X'000000B5'
MQCMD_INQUIRE_SUB_STATUS	182	X'000000B6'
MQCMD_INQUIRE_TOPIC_STATUS	183	X'000000B7'
MQCMD_CLEAR_TOPIC_STRING	184	X'000000B8'
MQCMD_INQUIRE_PUBSUB_STATUS	185	X'000000B9'
MQCMD_PURGE_CHANNEL	195	X'000000C3'

MQCMDI_* (Command format Command Information Values):

MQCMDI_CMDSCOPE_ACCEPTED	1	X'00000001'
MQCMDI_CMDSCOPE_GENERATED	2	X'00000002'
MQCMDI_CMDSCOPE_COMPLETED	3	X'00000003'
MQCMDI_QSG_DISP_COMPLETED	4	X'00000004'
MQCMDI_COMMAND_ACCEPTED	5	X'00000005'
MQCMDI_CLUSTER_REQUEST_QUEUED	6	X'00000006'
MQCMDI_CHANNEL_INIT_STARTED	7	X'00000007'
MQCMDI_RECOVER_STARTED	11	X'0000000B'
MQCMDI_BACKUP_STARTED	12	X'0000000C'
MQCMDI_RECOVER_COMPLETED	13	X'0000000D'
MQCMDI_SEC_TIMER_ZERO	14	X'0000000E'
MQCMDI_REFRESH_CONFIGURATION	16	X'00000010'
MQCMDI_SEC_SIGNOFF_ERROR	17	X'00000011'
MQCMDI_IMS_BRIDGE_SUSPENDED	18	X'00000012'
MQCMDI_DB2_SUSPENDED	19	X'00000013'
MQCMDI_DB2_OBSOLETE_MSGS	20	X'00000014'
MQCMDI_SEC_UPPERCASE	21	X'00000015'
MQCMDI_SEC_MIXEDCASE	22	X'00000016'

MQCMDL_* (Command Levels):

MQCMDL_LEVEL_1	100
MQCMDL_LEVEL_101	101
MQCMDL_LEVEL_110	110
MQCMDL_LEVEL_114	114
MQCMDL_LEVEL_120	120
MQCMDL_LEVEL_200	200
MQCMDL_LEVEL_201	201
MQCMDL_LEVEL_210	210
MQCMDL_LEVEL_211	211
MQCMDL_LEVEL_220	220
MQCMDL_LEVEL_221	221
MQCMDL_LEVEL_230	230
MQCMDL_LEVEL_320	320
MQCMDL_LEVEL_420	420
MQCMDL_LEVEL_500	500
MQCMDL_LEVEL_510	510
MQCMDL_LEVEL_520	520
MQCMDL_LEVEL_530	530
MQCMDL_LEVEL_531	531
MQCMDL_LEVEL_600	600
MQCMDL_LEVEL_700	700

MQCMDL_LEVEL_701	701
MQCMDL_LEVEL_710	710
MQCMDL_LEVEL_711	711
MQCMDL_LEVEL_750	750

MQCMHO_* (Create message handle options and structure):
Create message handle options structure

MQCMHO_STRUC_ID	"CMHO"	
MQCMHO_STRUC_ID_ARRAY	'C','M','H','O'	
MQCMHO_VERSION_1	1	X'00000001'
MQCMHO_CURRENT_VERSION	1	X'00000001'

Create Message Handle Options

MQCMHO_DEFAULT_VALIDATION	0	X'00000000'
MQCMHO_NO_VALIDATION	1	X'00000001'
MQCMHO_VALIDATE	2	X'00000002'
MQCMHO_NONE	0	X'00000000'

MQCNO_* (Connect options and structure):
Connect options structure

MQCNO_STRUC_ID	"CNOb"	
MQCNO_STRUC_ID_ARRAY	'C','N','O','b'	
MQCNO_VERSION_1	1	X'00000001'
MQCNO_VERSION_2	2	X'00000002'
MQCNO_VERSION_3	3	X'00000003'
MQCNO_VERSION_4	4	X'00000004'
MQCNO_VERSION_5	5	X'00000005'
MQCNO_CURRENT_VERSION	5	X'00000005'

Connect Options

MQCNO_STANDARD_BINDING	0	X'00000000'
MQCNO_FASTPATH_BINDING	1	X'00000001'
MQCNO_SERIALIZE_CONN_TAG_Q_MGR	2	X'00000002'
MQCNO_SERIALIZE_CONN_TAG_QSG	4	X'00000004'
MQCNO_RESTRICT_CONN_TAG_Q_MGR	8	X'00000008'
MQCNO_RESTRICT_CONN_TAG_QSG	16	X'00000010'
MQCNO_HANDLE_SHARE_NONE	32	X'00000020'
MQCNO_HANDLE_SHARE_BLOCK	64	X'00000040'
MQCNO_HANDLE_SHARE_NO_BLOCK	128	X'00000080'
MQCNO_SHARED_BINDING	256	X'00000100'

MQCNO_ISOLATED_BINDING	512	X'00000200'
MQCNO_LOCAL_BINDING	1024	X'00000400'
MQCNO_CLIENT_BINDING	2048	X'00000800'
MQCNO_ACCOUNTING_MQI_ENABLED	4096	X'00001000'
MQCNO_ACCOUNTING_MQI_DISABLED	8192	X'00002000'
MQCNO_ACCOUNTING_Q_ENABLED	16384	X'00004000'
MQCNO_ACCOUNTING_Q_DISABLED	32768	X'00008000'
MQCNO_NO_CONV_SHARING	65536	X'00010000'
MQCNO_ALL_CONVS_SHARE	262144	X'00040000'
MQCNO_CD_FOR_OUTPUT_ONLY	524288	X'00080000'
MQCNO_USE_CD_SELECTION	1048576	X'00100000'
MQCNO_RECONNECT_AS_DEF	0	X'00000000'
MQCNO_RECONNECT_DISABLED	33554432	X'02000000'
MQCNO_RECONNECT_Q_MGR	67108864	X'04000000'
MQCNO_ACTIVITY_TRACE_ENABLED	134217728	X'08000000'
MQCNO_ACTIVITY_TRACE_DISABLED	268435456	X'10000000'
MQCNO_NONE	0	X'00000000'

MQCO_* (Close Options):

MQCO_IMMEDIATE	0	X'00000000'
MQCO_NONE	0	X'00000000'
MQCO_DELETE	1	X'00000001'
MQCO_DELETE_PURGE	2	X'00000002'
MQCO_KEEP_SUB	4	X'00000004'
MQCO_REMOVE_SUB	8	X'00000008'
MQCO_QUIESCE	32	X'00000020'

MQCODL_* (CICS information header Output Data Length):

MQCODL_AS_INPUT	-1	X'FFFFFFFF'

MQCOMPRESS_* (Channel Compression):

MQCOMPRESS_NOT_AVAILABLE	-1	X'FFFFFFFF'
MQCOMPRESS_NONE	0	X'00000000'
MQCOMPRESS_RLE	1	X'00000001'
MQCOMPRESS_ZLIBFAST	2	X'00000002'
MQCOMPRESS_ZLIBHIGH	4	X'00000004'
MQCOMPRESS_SYSTEM	8	X'00000008'
MQCOMPRESS_ANY	268435455	X'0FFFFFFF'

MQCONNID_* (Connection Identifier):

MQCONNID_NONE	X'00...00'	(24 nulls)
MQCONNID_NONE_ARRAY	'\0','\0',...	(24 nulls)

MQCOPY_* (Property Copy Options):

MQCOPY_NONE	0	X'00000000'
MQCOPY_ALL	1	X'00000001'
MQCOPY_FORWARD	2	X'00000002'
MQCOPY_PUBLISH	4	X'00000004'
MQCOPY_REPLY	8	X'00000008'
MQCOPY_REPORT	16	X'00000010'
MQCOPY_DEFAULT	22	X'00000016'

MQCQT_* (Cluster Queue Types):

MQCQT_LOCAL_Q	1	X'00000001'
MQCQT_ALIAS_Q	2	X'00000002'
MQCQT_REMOTE_Q	3	X'00000003'
MQCQT_Q_MGR_ALIAS	4	X'00000004'

MQCRC_* (CICS information header Return Codes):

MQCRC_OK	0	X'00000000'
MQCRC_CICS_EXEC_ERROR	1	X'00000001'
MQCRC_MQ_API_ERROR	2	X'00000002'
MQCRC_BRIDGE_ERROR	3	X'00000003'
MQCRC_BRIDGE_ABEND	4	X'00000004'
MQCRC_APPLICATION_ABEND	5	X'00000005'
MQCRC_SECURITY_ERROR	6	X'00000006'
MQCRC_PROGRAM_NOT_AVAILABLE	7	X'00000007'
MQCRC_BRIDGE_TIMEOUT	8	X'00000008'
MQCRC_TRANSID_NOT_AVAILABLE	9	X'00000009'

MQCS_* (MQCBC constants Consumer state):

MQCS_NONE	0	X'00000000'
MQCS_SUSPENDED_TEMPORARY	1	X'00000001'
MQCS_SUSPENDED_USER_ACTION	2	X'00000002'
MQCS_SUSPENDED	3	X'00000003'
MQCS_STOPPED	4	X'00000004'

MQCSC_* (CICS information header Start Codes):

MQCSC_START	"Sbbb"	
MQCSC_STARTDATA	"SDbb"	
MQCSC_TERMINPUT	"TDbb"	
MQCSC_NONE	"bbbb"	
MQCSC_START_ARRAY	'S','b','b','b'	
MQCSC_STARTDATA_ARRAY	'S','D','b','b'	
MQCSC_TERMINPUT_ARRAY	'T','D','b','b'	
MQCSC_NONE_ARRAY	'b','b','b','b'	

MQCSP_* (Connection security parameters structure and Authentication Types):
Connection security parameters structure

MQCSP_STRUC_ID	"CSPb"	
MQCSP_STRUC_ID_ARRAY	'C','S','P','b'	
MQCSP_VERSION_1	1	X'00000001'
MQCSP_CURRENT_VERSION	1	X'00000001'

Connection security parameters Authentication Types

MQCSP_AUTH_NONE	0	X'00000000'
MQCSP_AUTH_USER_ID_AND_PWD	1	X'00000001'

MQCSRV_* (Command Server Options):

MQCSRV_CONVERT_NO	0	X'00000000'
MQCSRV_CONVERT_YES	1	X'00000001'
MQCSRV_DLQ_NO	0	X'00000000'
MQCSRV_DLQ_YES	1	X'00000001'

MQCT_* (Queue Manager Connection Tag):

MQCT_NONE	X'00...00'	(128 nulls)
MQCT_NONE_ARRAY	'\0','\0',...	(128 nulls)

MQCTES_* (CICS information header Task End Status):

MQCTES_NOSYNC	0	X'00000000'
MQCTES_COMMIT	256	X'00000100'
MQCTES_BACKOUT	4352	X'00001100'
MQCTES_ENDTASK	65536	X'00010000'

MQCTLO_* (MQCTL options structure and Consumer Control Options):
MQCTL options structure

MQCTLO_STRUC_ID	"CTLO"	
MQCTLO_STRUC_ID_ARRAY	'C','T','L','O'	
MQCTLO_VERSION_1	1	X'00000001'
MQCTLO_CURRENT_VERSION	1	X'00000001'

MQCTL options Consumer Control Options

MQCTLO_NONE	0	X'00000000'
MQCTLO_THREAD_AFFINITY	1	X'00000001'
MQCTLO_FAIL_IF_QUIESCING	8192	X'00002000'

MQCUOWC_* (CICS information header Unit-of-Work Controls):

MQCUOWC_ONLY	273	X'00000111'
MQCUOWC_CONTINUE	65536	X'00010000'
MQCUOWC_FIRST	17	X'00000011'
MQCUOWC_MIDDLE	16	X'00000010'
MQCUOWC_LAST	272	X'00000110'
MQCUOWC_COMMIT	256	X'00000100'
MQCUOWC_BACKOUT	4352	X'00001100'

MQCXP_* (Channel exit parameter structure):

MQCXP_STRUC_ID	"CXPb"	
MQCXP_STRUC_ID_ARRAY	'C','X','P','b'	
MQCXP_VERSION_1	1	X'00000001'
MQCXP_VERSION_2	2	X'00000002'
MQCXP_VERSION_3	3	X'00000003'
MQCXP_VERSION_4	4	X'00000004'
MQCXP_VERSION_5	5	X'00000005'
MQCXP_VERSION_6	6	X'00000006'
MQCXP_VERSION_7	7	X'00000007'
MQCXP_VERSION_8	8	X'00000008'
MQCXP_CURRENT_VERSION	8	X'00000008'

MQDC_* (Destination Class):

MQDC_MANAGED	1	X'00000001'
MQDC_PROVIDED	2	X'00000002'

MQDCC_* (Conversion Options, and Masks and Factors):
Conversion Options

MQDCC_DEFAULT_CONVERSION	1	X'00000001'
MQDCC_FILL_TARGET_BUFFER	2	X'00000002'
MQDCC_INT_DEFAULT_CONVERSION	4	X'00000004'
MQDCC_SOURCE_ENC_NATIVE	(value differs by platform or version)	
MQDCC_SOURCE_ENC_NORMAL	16	X'00000010'
MQDCC_SOURCE_ENC_REVERSED	32	X'00000020'
MQDCC_SOURCE_ENC_UNDEFINED	0	X'00000000'
MQDCC_TARGET_ENC_NATIVE	(value differs by platform or version)	
MQDCC_TARGET_ENC_NORMAL	256	X'00000100'
MQDCC_TARGET_ENC_REVERSED	512	X'00000200'
MQDCC_TARGET_ENC_UNDEFINED	0	X'00000000'
MQDCC_NONE	0	X'00000000'

Conversion Options Masks and Factors

MQDCC_SOURCE_ENC_MASK	240	X'000000F0'
MQDCC_TARGET_ENC_MASK	3840	X'00000F00'
MQDCC_SOURCE_ENC_FACTOR	16	X'00000010'
MQDCC_TARGET_ENC_FACTOR	256	X'00000100'

MQDELO_* (Publish/Subscribe Delete Options):

MQDELO_NONE	0	X'00000000'
MQDELO_LOCAL	4	X'00000004'

MQDH_* (Distribution header structure):

MQDH_STRUC_ID	"DHbb"	
MQDH_STRUC_ID_ARRAY	'D','H','b','b'	
MQDH_VERSION_1	1	X'00000001'
MQDH_CURRENT_VERSION	1	X'00000001'

MQDHF_* (Distribution header Flags):

MQDHF_NEW_MSG_IDS	1	X'00000001'
MQDHF_NONE	0	X'00000000'

MQDISCONNECT_* (Command format Disconnect Types):

MQDISCONNECT_NORMAL	0	X'00000000'
MQDISCONNECT_IMPLICIT	1	X'00000001'
MQDISCONNECT_Q_MGR	2	X'00000002'

MQDL_* (Distribution Lists):

MQDL_SUPPORTED	1	X'00000001'
MQDL_NOT_SUPPORTED	0	X'00000000'

MQDLH_* (Dead-letter header structure):

MQDLH_STRUC_ID	"DLHb"	
MQDLH_STRUC_ID_ARRAY	'D','L','H','b'	
MQDLH_VERSION_1	1	X'00000001'
MQDLH_CURRENT_VERSION	1	X'00000001'

MQDLV_* (Persistent/Non-persistent Message Delivery):

MQDLV_AS_PARENT	0	X'00000000'
MQDLV_ALL	1	X'00000001'
MQDLV_ALL_DUR	2	X'00000002'
MQDLV_ALL_AVAIL	3	X'00000003'

MQDMHO_* (Delete message handle options and structure):
Delete message handle options structure

MQDMHO_STRUC_ID	"DMHO"	
MQDMHO_STRUC_ID_ARRAY	'D','M','H','O'	
MQDMHO_VERSION_1	1	X'00000001'
MQDMHO_CURRENT_VERSION	1	X'00000001'

Delete Message Handle Options

MQDMHO_NONE	0	X'00000000'

MQDMPO_* (Delete message property options and structure):
Delete message property options structure

MQDMPO_STRUC_ID	"DMPO"	
MQDMPO_STRUC_ID_ARRAY	'D','M','P','O'	
MQDMPO_VERSION_1	1	X'00000001'
MQDMPO_CURRENT_VERSION	1	X'00000001'

Delete Message Property Options

MQDMPO_DEL_FIRST	0	X'00000000'
MQDMPO_DEL_PROP_UNDER_CURSOR	1	X'00000001'
MQDMPO_NONE	0	X'00000000'

MQDNSWLM_* (DNS WLM):

MQDNSWLM_NO	0	X'00000000'
MQDNSWLM_YES	1	X'00000001'

MQDT_* (Destination Types):

MQDT_APPL	1	X'00000001'
MQDT_BROKER	2	X'00000002'

MQDXP_* (Conversion exit parameter structure):

MQDXP_STRUC_ID	"DXPb"	
MQDXP_STRUC_ID_ARRAY	'D','X','P','b'	
MQDXP_VERSION_1	1	X'00000001'
MQDXP_VERSION_2	2	X'00000002'
MQDXP_CURRENT_VERSION	2	X'00000002'

MQEC_* (Signal Values):

MQEC_MSG_ARRIVED	2	X'00000002'
MQEC_WAIT_INTERVAL_EXPIRED	3	X'00000003'
MQEC_WAIT_CANCELED	4	X'00000004'
MQEC_Q_MGR_QUIESCING	5	X'00000005'
MQEC_CONNECTION_QUIESCING	6	X'00000006'

MQEI_* (Expiry):

MQEI_UNLIMITED	-1	X'FFFFFFFF'

MQENC_* (Encoding):
MQENC_* (Encoding)

MQENC_NATIVE	IBM i	273	X'00000111'
	Linux	546	X'00000222'
	Linux on SPARC	273	X'00000111'
	Linux on x86	546	X'00000222'
	Solaris on SPARC	273	X'00000111'
	UNIX systems	273	X'00000111'
	Windows	546	X'00000222'
	Micro Focus COBOL on Windows	17	X'00000011'
	z/OS	785	X'00000311'

MQENC_* (Encoding Masks)

MQENC_INTEGER_MASK	15	X'0000000F'
MQENC_DECIMAL_MASK	240	X'000000F0'
MQENC_FLOAT_MASK	3840	X'00000F00'
MQENC_RESERVED_MASK	-4096	X'FFFFF000'

MQENC_* (Encodings for Binary Integers)

MQENC_INTEGER_UNDEFINED	0	X'00000000'
MQENC_INTEGER_NORMAL	1	X'00000001'
MQENC_INTEGER_REVERSED	2	X'00000002'

MQENC_* (Encodings for Packed Decimal Integers)

MQENC_DECIMAL_UNDEFINED	0	X'00000000'
MQENC_DECIMAL_NORMAL	16	X'00000010'
MQENC_DECIMAL_REVERSED	32	X'00000020'

MQENC_* (Encodings for Floating Point Numbers)

MQENC_FLOAT_UNDEFINED	0	X'00000000'
MQENC_FLOAT_IEEE_NORMAL	256	X'00000100'
MQENC_FLOAT_IEEE_REVERSED	512	X'00000200'
MQENC_FLOAT_S390	768	X'00000300'
MQENC_FLOAT_TNS	1024	X'00000400'

MQEPH_* (Embedded command format header structure and Flags):
Embedded command format header structure

MQEPH_STRUC_ID	"EPHb"	
MQEPH_STRUC_ID_ARRAY	'E','P','H','b'	
MQEPH_STRUC_LENGTH_FIXED	68	X'00000044'
MQEPH_VERSION_1	1	X'00000001'
MQEPH_CURRENT_VERSION	1	X'00000001'

Embedded command format header Flags

MQEPH_NONE	0	X'00000000'
MQEPH_CCSID_EMBEDDED	1	X'00000001'

MQET_* (Command format Escape Types):

MQET_MQSC	1	X'00000001'

MQEVO_* (Command format Event Origins):

MQEVO_OTHER	0	X'00000000'
MQEVO_CONSOLE	1	X'00000001'
MQEVO_INIT	2	X'00000002'
MQEVO_MSG	3	X'00000003'
MQEVO_MQSET	4	X'00000004'
MQEVO_INTERNAL	5	X'00000005'

MQEVR_* (Command format Event Recording):

MQEVR_DISABLED	0	X'00000000'
MQEVR_ENABLED	1	X'00000001'
MQEVR_EXCEPTION	2	X'00000002'
MQEVR_NO_DISPLAY	3	X'00000003'

MQEXPI_* (Expiration Scan Interval):

MQEXPI_OFF	0	X'00000000'

MQFB_* (Feedback Values):

MQFB_NONE	0	X'00000000'
MQFB_SYSTEM_FIRST	1	X'00000001'
MQFB_QUIT	256	X'00000100'
MQFB_EXPIRATION	258	X'00000102'
MQFB_COA	259	X'00000103'
MQFB_COD	260	X'00000104'
MQFB_CHANNEL_COMPLETED	262	X'00000106'
MQFB_CHANNEL_FAIL_RETRY	263	X'00000107'

MQFB_CHANNEL_FAIL	264	X'00000108'
MQFB_APPL_CANNOT_BE_STARTED	265	X'00000109'
MQFB_TM_ERROR	266	X'0000010A'
MQFB_APPL_TYPE_ERROR	267	X'0000010B'
MQFB_STOPPED_BY_MSG_EXIT	268	X'0000010C'
MQFB_ACTIVITY	269	X'0000010D'
MQFB_XMIT_Q_MSG_ERROR	271	X'0000010F'
MQFB_PAN	275	X'00000113'
MQFB_NAN	276	X'00000114'
MQFB_STOPPED_BY_CHAD_EXIT	277	X'00000115'
MQFB_STOPPED_BY_PUBSUB_EXIT	279	X'00000117'
MQFB_NOT_A_REPOSITORY_MSG	280	X'00000118'
MQFB_BIND_OPEN_CLUSRCVR_DEL	281	X'00000119'
MQFB_MAX_ACTIVITIES	282	X'0000011A'
MQFB_NOT_FORWARDED	283	X'0000011B'
MQFB_NOT_DELIVERED	284	X'0000011C'
MQFB_UNSUPPORTED_FORWARDING	285	X'0000011D'
MQFB_UNSUPPORTED_DELIVERY	286	X'0000011E'
MQFB_DATA_LENGTH_ZERO	291	X'00000123'
MQFB_DATA_LENGTH_NEGATIVE	292	X'00000124'
MQFB_DATA_LENGTH_TOO_BIG	293	X'00000125'
MQFB_BUFFER_OVERFLOW	294	X'00000126'
MQFB_LENGTH_OFF_BY_ONE	295	X'00000127'
MQFB_IIH_ERROR	296	X'00000128'
MQFB_NOT_AUTHORIZED_FOR_IMS	298	X'0000012A'
MQFB_IMS_ERROR	300	X'0000012C'
MQFB_IMS_FIRST	301	X'0000012D'
MQFB_IMS_LAST	399	X'0000018F'
MQFB_CICS_INTERNAL_ERROR	401	X'00000191'
MQFB_CICS_NOT_AUTHORIZED	402	X'00000192'
MQFB_CICS_BRIDGE_FAILURE	403	X'00000193'
MQFB_CICS_CORREL_ID_ERROR	404	X'00000194'
MQFB_CICS_CCSID_ERROR	405	X'00000195'
MQFB_CICS_ENCODING_ERROR	406	X'00000196'
MQFB_CICS_CIH_ERROR	407	X'00000197'
MQFB_CICS_UOW_ERROR	408	X'00000198'
MQFB_CICS_COMMAREA_ERROR	409	X'00000199'
MQFB_CICS_APPL_NOT_STARTED	410	X'0000019A'
MQFB_CICS_APPL_ABENDED	411	X'0000019B'
MQFB_CICS_DLQ_ERROR	412	X'0000019C'
MQFB_CICS_UOW_BACKED_OUT	413	X'0000019D'
MQFB_PUBLICATIONS_ON_REQUEST	501	X'000001F5'

MQFB_SUBSCRIBER_IS_PUBLISHER	502	X'000001F6'
MQFB_MSG_SCOPE_MISMATCH	503	X'000001F7'
MQFB_SELECTOR_MISMATCH	504	X'000001F8'
MQFB_IMS_NACK_1A_REASON_FIRST	600	X'00000258'
MQFB_IMS_NACK_1A_REASON_LAST	855	X'00000357'
MQFB_SYSTEM_LAST	65535	X'0000FFFF'
MQFB_APPL_FIRST	65536	X'00010000'
MQFB_APPL_LAST	999999999	X'3B9AC9FF'

MQFC_* (Command format Force Options):

MQFC_YES	1	X'00000001'
MQFC_NO	0	X'00000000'

MQFMT_* (Formats):

MQFMT_NONE	"bbbbbbbb"
MQFMT_ADMIN	"MQADMINb"
MQFMT_CHANNEL_COMPLETED	"MQCHCOMb"
MQFMT_CICS	"MQCICSbb"
MQFMT_COMMAND_1	"MQCMD1bb"
MQFMT_COMMAND_2	"MQCMD2bb"
MQFMT_DEAD_LETTER_HEADER	"MQDEADbb"
MQFMT_DIST_HEADER	"MQHDISTb"
MQFMT_EMBEDDED_PCF	"MQHEPCFb"
MQFMT_EVENT	"MQEVENTb"
MQFMT_IMS	"MQIMSbbb"
MQFMT_IMS_VAR_STRING	"MQIMSVSb"
MQFMT_MD_EXTENSION	"MQHMDEbb"
MQFMT_PCF	"MQPCFbbb"
MQFMT_REF_MSG_HEADER	"MQHREFbb"
MQFMT_RF_HEADER	"MQHRFbbb"
MQFMT_RF_HEADER_1	"MQHRFbbb"
MQFMT_RF_HEADER_2	"MQHRF2bb"
MQFMT_STRING	"MQSTRbbb"
MQFMT_TRIGGER	"MQTRIGbb"
MQFMT_WORK_INFO_HEADER	"MQHWIHbb"
MQFMT_XMIT_Q_HEADER	"MQXMITbb"
MQFMT_NONE_ARRAY	'b','b','b','b','b','b','b','b'
MQFMT_ADMIN_ARRAY	'M','Q','A','D','M','I','N','b'
MQFMT_CHANNEL_COMPLETED_ARRAY	'M','Q','C','H','C','O','M','b'
MQFMT_CICS_ARRAY	'M','Q','C','I','C','S','b','b'
MQFMT_COMMAND_1_ARRAY	'M','Q','C','M','D','1','b','b'

MQFMT_COMMAND_2_ARRAY	'M','Q','C','M','D','2','b','b'
MQFMT_DEAD_LETTER_HEADER_ARRAY	'M','Q','D','E','A','D','b','b'
MQFMT_DIST_HEADER_ARRAY	'M','Q','H','D','I','S','T','b'
MQFMT_EMBEDDED_PCF_ARRAY	'M','Q','H','E','P','C','F','b'
MQFMT_EVENT_ARRAY	'M','Q','E','V','E','N','T','b'
MQFMT_IMS_ARRAY	'M','Q','I','M','S','b','b','b'
MQFMT_IMS_VAR_STRING_ARRAY	'M','Q','I','M','S','V','S','b'
MQFMT_MD_EXTENSION_ARRAY	'M','Q','H','M','D','E','b','b'
MQFMT_PCF_ARRAY	'M','Q','P','C','F','b','b','b'
MQFMT_REF_MSG_HEADER_ARRAY	'M','Q','H','R','E','F','b','b'
MQFMT_RF_HEADER_ARRAY	'M','Q','H','R','F','b','b','b'
MQFMT_RF_HEADER_1_ARRAY	'M','Q','H','R','F','b','b','b'
MQFMT_RF_HEADER_2_ARRAY	'M','Q','H','R','F','2','b','b'
MQFMT_STRING_ARRAY	'M','Q','S','T','R','b','b','b'
MQFMT_TRIGGER_ARRAY	'M','Q','T','R','I','G','b','b'
MQFMT_WORK_INFO_HEADER_ARRAY	'M','Q','H','W','I','H','b','b'
MQFMT_XMIT_Q_HEADER_ARRAY	'M','Q','X','M','I','T','b','b'

MQGA_* (Group Attribute Selectors):

MQGA_FIRST	8001	X'00001F41'
MQGA_LAST	9000	X'00002328'

MQGACF_* (Command format Group Parameter Types):

MQGACF_FIRST	8001	X'00001F41'
MQGACF_COMMAND_CONTEXT	8001	X'00001F41'
MQGACF_COMMAND_DATA	8002	X'00001F42'
MQGACF_TRACE_ROUTE	8003	X'00001F43'
MQGACF_OPERATION	8004	X'00001F44'
MQGACF_ACTIVITY	8005	X'00001F45'
MQGACF_EMBEDDED_MQMD	8006	X'00001F46'
MQGACF_MESSAGE	8007	X'00001F47'
MQGACF_MQMD	8008	X'00001F48'
MQGACF_VALUE_NAMING	8009	X'00001F49'
MQGACF_Q_ACCOUNTING_DATA	8010	X'00001F4A'
MQGACF_Q_STATISTICS_DATA	8011	X'00001F4B'
MQGACF_CHL_STATISTICS_DATA	8012	X'00001F4C'
MQGACF_LAST_USED	8012	X'00001F4C'

MQGI_* (Group Identifier):

MQGI_NONE	X'00...00'	(24 nulls)
MQGI_NONE_ARRAY	'\0','\0',...	(24 nulls)

MQGMO_* (Get message options and structure):
Get message options structure

MQGMO_STRUC_ID	"GMOb"	
MQGMO_STRUC_ID_ARRAY	'G','M','O','b'	
MQGMO_VERSION_1	1	X'00000001'
MQGMO_VERSION_2	2	X'00000002'
MQGMO_VERSION_3	3	X'00000003'
MQGMO_VERSION_4	4	X'00000004'
MQGMO_CURRENT_VERSION	4	X'00000004'

Get Message Options

MQGMO_WAIT	1	X'00000001'
MQGMO_NO_WAIT	0	X'00000000'
MQGMO_SET_SIGNAL	8	X'00000008'
MQGMO_FAIL_IF_QUIESCING	8192	X'00002000'
MQGMO_SYNCPOINT	2	X'00000002'
MQGMO_SYNCPOINT_IF_PERSISTENT	4096	X'00001000'
MQGMO_NO_SYNCPOINT	4	X'00000004'
MQGMO_MARK_SKIP_BACKOUT	128	X'00000080'
MQGMO_BROWSE_FIRST	16	X'00000010'
MQGMO_BROWSE_NEXT	32	X'00000020'
MQGMO_BROWSE_MSG_UNDER_CURSOR	2048	X'00000800'
MQGMO_BROWSE_HANDLE	17825808	X'01100010'
MQGMO_BROWSE_CO_OP	18874384	X'01200010'
MQGMO_MSG_UNDER_CURSOR	256	X'00000100'
MQGMO_LOCK	512	X'00000200'
MQGMO_UNLOCK	1024	X'00000400'
MQGMO_ACCEPT_TRUNCATED_MSG	64	X'00000040'
MQGMO_CONVERT	16384	X'00004000'
MQGMO_LOGICAL_ORDER	32768	X'00008000'
MQGMO_COMPLETE_MSG	65536	X'00010000'
MQGMO_ALL_MSGS_AVAILABLE	131072	X'00020000'
MQGMO_ALL_SEGMENTS_AVAILABLE	262144	X'00040000'
MQGMO_MARK_BROWSE_HANDLE	1048576	X'00100000'
MQGMO_MARK_BROWSE_CO_OP	2097152	X'00200000'
MQGMO_UNMARK_BROWSE_CO_OP	4194304	X'00400000'
MQGMO_UNMARK_BROWSE_HANDLE	8388608	X'00800000'
MQGMO_UNMARKED_BROWSE_MSG	16777216	X'01000000'

MQGMO_PROPERTIES_FORCE_MQRFH2	33554432	X'02000000'
MQGMO_NO_PROPERTIES	67108864	X'04000000'
MQGMO_PROPERTIES_IN_HANDLE	134217728	X'08000000'
MQGMO_PROPERTIES_COMPATIBILITY	268435456	X'10000000'
MQGMO_PROPERTIES_AS_Q_DEF	0	X'00000000'
MQGMO_NONE	0	X'00000000'

MQGS_* (Group Status):

MQGS_NOT_IN_GROUP	'b' (blank)	
MQGS_MSG_IN_GROUP	'G'	
MQGS_LAST_MSG_IN_GROUP	'L'	

MQHA_* (Handle Selectors):

MQHA_FIRST	4001	X'00000FA1'
MQHA_BAG_HANDLE	4001	X'00000FA1'
MQHA_LAST_USED	4001	X'00000FA1'
MQHA_LAST	6000	X'00001770'

MQHB_* (Bag Handles):

MQHB_UNUSABLE_HBAG	-1	X'FFFFFFFF'
MQHB_NONE	-2	X'FFFFFFFE'

MQHC_* (Connection Handles):

MQHC_DEF_HCONN	0	X'00000000'
MQHC_UNUSABLE_HCONN	-1	X'FFFFFFFF'
MQHC_UNASSOCIATED_HCONN	-3	X'FFFFFFFD'

MQHM_* (Message handle):

MQHM_UNUSABLE_HMSG	-1	X'FFFFFFFF'
MQHM_NONE	0	X'00000000'

MQHO_* (Object Handle):

MQHO_UNUSABLE_HOBJ	-1	X'FFFFFFFF'
MQHO_NONE	0	X'00000000'

MQHSTATE_* (Command format Handle States):

MQHSTATE_INACTIVE	0	X'00000000'
MQHSTATE_ACTIVE	1	X'00000001'

MQIA_* (Integer Attribute Selectors):

MQIA_ACCOUNTING_CONN_OVERRIDE	136	X'00000088'
MQIA_ACCOUNTING_INTERVAL	135	X'00000087'
MQIA_ACCOUNTING_MQI	133	X'00000085'
MQIA_ACCOUNTING_Q	134	X'00000086'
MQIA_ACTIVE_CHANNELS	100	X'00000064'
MQIA_ACTIVITY_CONN_OVERRIDE	239	X'000000EF'
MQIA_ACTIVITY_RECORDING	138	X'0000008A'
MQIA_ACTIVITY_TRACE	240	X'000000F0'
MQIA_ADOPTNEWMCA_CHECK	102	X'00000066'
MQIA_ADOPTNEWMCA_TYPE	103	X'00000067'
MQIA_ADOPTNEWMCA_INTERVAL	104	X'00000068'
MQIA_APPL_TYPE	1	X'00000001'
MQIA_ARCHIVE	60	X'0000003C'
MQIA_AUTH_INFO_TYPE	66	X'00000042'
MQIA_AUTHORITY_EVENT	47	X'0000002F'
MQIA_AUTO_REORG_INTERVAL	174	X'000000AE'
MQIA_AUTO_REORGANIZATION	173	X'000000AD'
MQIA_BACKOUT_THRESHOLD	22	X'00000016'
MQIA_BASE_TYPE	193	X'000000C1'
MQIA_BATCH_INTERFACE_AUTO	86	X'00000056'
MQIA_BRIDGE_EVENT	74	X'0000004A'
MQIA_CF_LEVEL	70	X'00000046'
MQIA_CF_RECOVER	71	X'00000047'
MQIA_CHANNEL_AUTO_DEF	55	X'00000037'
MQIA_CHANNEL_AUTO_DEF_EVENT	56	X'00000038'
MQIA_CHANNEL_EVENT	73	X'00000049'
MQIA_CHINIT_ADAPTERS	101	X'00000065'
MQIA_CHINIT_CONTROL	119	X'00000077'
MQIA_CHINIT_DISPATCHERS	105	X'00000069'
MQIA_CHINIT_TRACE_AUTO_START	117	X'00000075'
MQIA_CHINIT_TRACE_TABLE_SIZE	118	X'00000076'
MQIA_CLUSTER_Q_TYPE	59	X'0000003B'
MQIA_CLUSTER_WORKLOAD_LENGTH	58	X'0000003A'

MQIA_CLWL_MRU_CHANNELS	97	X'00000061'
MQIA_CLWL_Q_RANK	95	X'0000005F'
MQIA_CLWL_Q_PRIORITY	96	X'00000060'
MQIA_CLWL_USEQ	98	X'00000062'
MQIA_CMD_SERVER_AUTO	87	X'00000057'
MQIA_CMD_SERVER_CONTROL	120	X'00000078'
MQIA_CMD_SERVER_CONVERT_MSG	88	X'00000058'
MQIA_CMD_SERVER_DLQ_MSG	89	X'00000059'
MQIA_CODED_CHAR_SET_ID	2	X'00000002'
MQIA_COMM_EVENT	232	X'000000E8'
MQIA_COMMAND_EVENT	99	X'00000063'
MQIA_COMMAND_LEVEL	31	X'0000001F'
MQIA_CONFIGURATION_EVENT	51	X'00000033'
MQIA_CPI_LEVEL	27	X'0000001B'
MQIA_CURRENT_Q_DEPTH	3	X'00000003'
MQIA_DEF_BIND	61	X'0000003D'
MQIA_DEF_CLUSTER_XMIT_Q_TYPE	250	X'000000FA'
MQIA_DEF_INPUT_OPEN_OPTION	4	X'00000004'
MQIA_DEF_PERSISTENCE	5	X'00000005'
MQIA_DEF_PRIORITY	6	X'00000006'
MQIA_DEF_PUT_RESPONSE_TYPE	184	X'000000B8'
MQIA_DEF_READ_AHEAD	188	X'000000BC'
MQIA_DEFINITION_TYPE	7	X'00000007'
MQIA_DIST_LISTS	34	X'00000022'
MQIA_DNS_WLM	106	X'0000006A'
MQIA_DURABLE_SUB	175	X'000000AF'
MQIA_EXPIRY_INTERVAL	39	X'00000027'
MQIA_FIRST	1	X'00000001'
MQIA_HARDEN_GET_BACKOUT	8	X'00000008'
MQIA_HIGH_Q_DEPTH	36	X'00000024'
MQIA_IGQ_PUT_AUTHORITY	65	X'00000041'
MQIA_INDEX_TYPE	57	X'00000039'
MQIA_INHIBIT_EVENT	48	X'00000030'
MQIA_INHIBIT_GET	9	X'00000009'
MQIA_INHIBIT_PUB	181	X'000000B5'
MQIA_INHIBIT_PUT	10	X'0000000A'
MQIA_INHIBIT_SUB	182	X'000000B6'
MQIA_INTRA_GROUP_QUEUING	64	X'00000040'
MQIA_IP_ADDRESS_VERSION	93	X'0000005D'
MQIA_LAST	2000	X'000007D0'
MQIA_LAST_USED	233	X'000000E9'
MQIA_LISTENER_PORT_NUMBER	85	X'00000055'

MQIA_LISTENER_TIMER	107	X'0000006B'
MQIA_LOGGER_EVENT	94	X'0000005E'
MQIA_LU62_CHANNELS	108	X'0000006C'
MQIA_LOCAL_EVENT	49	X'00000031'
MQIA_MSG_MARK_BROWSE_INTERVAL	68	X'00000044'
MQIA_MAX_CHANNELS	109	X'0000006D'
MQIA_MAX_CLIENTS	172	X'000000AC'
MQIA_MAX_GLOBAL_LOCKS	83	X'00000053'
MQIA_MAX_HANDLES	11	X'0000000B'
MQIA_MAX_LOCAL_LOCKS	84	X'00000054'
MQIA_MAX_MSG_LENGTH	13	X'0000000D'
MQIA_MAX_OPEN_Q	80	X'00000050'
MQIA_MAX_PRIORITY	14	X'0000000E'
MQIA_MAX_PROPERTIES_LENGTH	192	X'000000C0'
MQIA_MAX_Q_DEPTH	15	X'0000000F'
MQIA_MAX_Q_TRIGGERS	90	X'0000005A'
MQIA_MAX_RECOVERY_TASKS	171	X'000000AB'
MQIA_MAX_UNCOMMITTED_MSGS	33	X'00000021'
MQIA_MCAST_BRIDGE	233	X'000000E9'
MQIA_MONITOR_INTERVAL	81	X'00000051'
MQIA_MONITORING_AUTO_CLUSSDR	124	X'0000007C'
MQIA_MONITORING_CHANNEL	122	X'0000007A'
MQIA_MONITORING_Q	123	X'0000007B'
MQIA_MSG_DELIVERY_SEQUENCE	16	X'00000010'
MQIA_MSG_DEQ_COUNT	38	X'00000026'
MQIA_MSG_ENQ_COUNT	37	X'00000025'
MQIA_NAME_COUNT	19	X'00000013'
MQIA_NAMELIST_TYPE	72	X'00000048'
MQIA_NPM_CLASS	78	X'0000004E'
MQIA_NPM_DELIVERY	196	X'000000C4'
MQIA_OPEN_INPUT_COUNT	17	X'00000011'
MQIA_OPEN_OUTPUT_COUNT	18	X'00000012'
MQIA_OUTBOUND_PORT_MAX	140	X'0000008C'
MQIA_OUTBOUND_PORT_MIN	110	X'0000006E'
MQIA_PAGESET_ID	62	X'0000003E'
MQIA_PERFORMANCE_EVENT	53	X'00000035'
MQIA_PLATFORM	32	X'00000020'
MQIA_PM_DELIVERY	195	X'000000C3'
MQIA_PROPERTY_CONTROL	190	X'000000BE'
MQIA_PROT_POLICY_CAPABILITY	251	X'000000FB'
MQIA_PROXY_SUB	199	X'000000C7'
MQIA_PUB_COUNT	215	X'000000D7'

MQIA_PUB_SCOPE	219	X'000000DB'
MQIA_PUBSUB_MAXMSG_RETRY_COUNT	206	X'000000CE'
MQIA_PUBSUB_MODE	187	X'000000BB'
MQIA_PUBSUB_NP_MSG	203	X'000000CB'
MQIA_PUBSUB_NP_RESP	205	X'000000CD'
MQIA_PUBSUB_SYNC_PT	207	X'000000CF'
MQIA_Q_DEPTH_HIGH_EVENT	43	X'0000002B'
MQIA_Q_DEPTH_HIGH_LIMIT	40	X'00000028'
MQIA_Q_DEPTH_LOW_EVENT	44	X'0000002C'
MQIA_Q_DEPTH_LOW_LIMIT	41	X'00000029'
MQIA_Q_DEPTH_MAX_EVENT	42	X'0000002A'
MQIA_Q_SERVICE_INTERVAL	54	X'00000036'
MQIA_Q_SERVICE_INTERVAL_EVENT	46	X'0000002E'
MQIA_Q_TYPE	20	X'00000014'
MQIA_Q_USERS	82	X'00000052'
MQIA_QMOPT_CONS_COMMS_MSGS	155	X'0000009B'
MQIA_QMOPT_CONS_CRITICAL_MSGS	154	X'0000009A'
MQIA_QMOPT_CONS_ERROR_MSGS	153	X'00000099'
MQIA_QMOPT_CONS_INFO_MSGS	151	X'00000097'
MQIA_QMOPT_CONS_REORG_MSGS	156	X'0000009C'
MQIA_QMOPT_CONS_SYSTEM_MSGS	157	X'0000009D'
MQIA_QMOPT_CONS_WARNING_MSGS	152	X'00000098'
MQIA_QMOPT_CSMT_ON_ERROR	150	X'00000096'
MQIA_QMOPT_INTERNAL_DUMP	170	X'000000AA'
MQIA_QMOPT_LOG_COMMS_MSGS	162	X'000000A2'
MQIA_QMOPT_LOG_CRITICAL_MSGS	161	X'000000A1'
MQIA_QMOPT_LOG_ERROR_MSGS	160	X'000000A0'
MQIA_QMOPT_LOG_INFO_MSGS	158	X'0000009E'
MQIA_QMOPT_LOG_REORG_MSGS	163	X'000000A3'
MQIA_QMOPT_LOG_SYSTEM_MSGS	164	X'000000A4'
MQIA_QMOPT_LOG_WARNING_MSGS	159	X'0000009F'
MQIA_QMOPT_TRACE_COMMS	166	X'000000A6'
MQIA_QMOPT_TRACE_CONVERSION	168	X'000000A8'
MQIA_QMOPT_TRACE_REORG	167	X'000000A7'
MQIA_QMOPT_TRACE_MQI_CALLS	165	X'000000A5'
MQIA_QMOPT_TRACE_SYSTEM	169	X'000000A9'
MQIA_QSG_DISP	63	X'0000003F'
MQIA_READ_AHEAD	189	X'000000BD'
MQIA_RECEIVE_TIMEOUT	111	X'0000006F'
MQIA_RECEIVE_TIMEOUT_MIN	113	X'00000071'
MQIA_RECEIVE_TIMEOUT_TYPE	112	X'00000070'
MQIA_REMOTE_EVENT	50	X'00000032'

MQIA_RETENTION_INTERVAL	21	X'00000015'
MQIA_SCOPE	45	X'0000002D'
MQIA_SECURITY_CASE	141	X'0000008D'
MQIA_SERVICE_CONTROL	139	X'0000008B'
MQIA_SERVICE_TYPE	121	X'00000079'
MQIA_SHAREABILITY	23	X'00000017'
MQIA_SHARED_Q_Q_MGR_NAME	77	X'0000004D'
MQIA_SSL_EVENT	75	X'0000004B'
MQIA_SSL_FIPS_REQUIRED	92	X'0000005C'
MQIA_SSL_RESET_COUNT	76	X'0000004C'
MQIA_SSL_TASKS	69	X'00000045'
MQIA_START_STOP_EVENT	52	X'00000034'
MQIA_STATISTICS_CHANNEL	129	X'00000081'
MQIA_STATISTICS_AUTO_CLUSSDR	130	X'00000082'
MQIA_STATISTICS_INTERVAL	131	X'00000083'
MQIA_STATISTICS_MQI	127	X'0000007F'
MQIA_STATISTICS_Q	128	X'00000080'
MQIA_SUB_COUNT	204	X'000000CC'
MQIA_SUB_SCOPE	218	X'000000DA'
MQIA_SYNCPOINT	30	X'0000001E'
MQIA_TCP_CHANNELS	114	X'00000072'
MQIA_TCP_KEEP_ALIVE	115	X'00000073'
MQIA_TCP_STACK_TYPE	116	X'00000074'
MQIA_TIME_SINCE_RESET	35	X'00000023'
MQIA_TOPIC_DEF_PERSISTENCE	185	X'000000B9'
MQIA_TOPIC_TYPE	208	X'000000D0'
MQIA_TRACE_ROUTE_RECORDING	137	X'00000089'
MQIA_TREE_LIFE_TIME	183	X'000000B7'
MQIA_TRIGGER_CONTROL	24	X'00000018'
MQIA_TRIGGER_DEPTH	29	X'0000001D'
MQIA_TRIGGER_INTERVAL	25	X'00000019'
MQIA_TRIGGER_MSG_PRIORITY	26	X'0000001A'
MQIA_TRIGGER_TYPE	28	X'0000001C'
MQIA_TRIGGER_RESTART	91	X'0000005B'
MQIA_USAGE	12	X'0000000C'
MQIA_USE_DEAD_LETTER_Q	234	X'000000EA'
MQIA_USER_LIST	2000	X'000007D0'
MQIA_WILDCARD_OPERATION	216	X'000000D8'
MQIA_XR_CAPABILITY	243	X'000000F3'

MQIACF_* (Command format Integer Parameter Types):

MQIACF_FIRST	1001	X'000003E9'
MQIACF_Q_MGR_ATTRS	1001	X'000003E9'
MQIACF_Q_ATTRS	1002	X'000003EA'
MQIACF_PROCESS_ATTRS	1003	X'000003EB'
MQIACF_NAMELIST_ATTRS	1004	X'000003EC'
MQIACF_FORCE	1005	X'000003ED'
MQIACF_REPLACE	1006	X'000003EE'
MQIACF_PURGE	1007	X'000003EF'
MQIACF_QUIESCE	1008	X'000003F0'
MQIACF_MODE	1008	X'000003F0'
MQIACF_ALL	1009	X'000003F1'
MQIACF_EVENT_APPL_TYPE	1010	X'000003F2'
MQIACF_EVENT_ORIGIN	1011	X'000003F3'
MQIACF_PARAMETER_ID	1012	X'000003F4'
MQIACF_ERROR_ID	1013	X'000003F5'
MQIACF_ERROR_IDENTIFIER	1013	X'000003F5'
MQIACF_SELECTOR	1014	X'000003F6'
MQIACF_CHANNEL_ATTRS	1015	X'000003F7'
MQIACF_OBJECT_TYPE	1016	X'000003F8'
MQIACF_ESCAPE_TYPE	1017	X'000003F9'
MQIACF_ERROR_OFFSET	1018	X'000003FA'
MQIACF_AUTH_INFO_ATTRS	1019	X'000003FB'
MQIACF_REASON_QUALIFIER	1020	X'000003FC'
MQIACF_COMMAND	1021	X'000003FD'
MQIACF_OPEN_OPTIONS	1022	X'000003FE'
MQIACF_OPEN_TYPE	1023	X'000003FF'
MQIACF_PROCESS_ID	1024	X'00000400'
MQIACF_THREAD_ID	1025	X'00000401'
MQIACF_Q_STATUS_ATTRS	1026	X'00000402'
MQIACF_UNCOMMITTED_MSGS	1027	X'00000403'
MQIACF_HANDLE_STATE	1028	X'00000404'
MQIACF_AUX_ERROR_DATA_INT_1	1070	X'0000042E'
MQIACF_AUX_ERROR_DATA_INT_2	1071	X'0000042F'
MQIACF_CONV_REASON_CODE	1072	X'00000430'
MQIACF_BRIDGE_TYPE	1073	X'00000431'
MQIACF_INQUIRY	1074	X'00000432'
MQIACF_WAIT_INTERVAL	1075	X'00000433'
MQIACF_OPTIONS	1076	X'00000434'
MQIACF_BROKER_OPTIONS	1077	X'00000435'
MQIACF_REFRESH_TYPE	1078	X'00000436'
MQIACF_SEQUENCE_NUMBER	1079	X'00000437'
MQIACF_INTEGER_DATA	1080	X'00000438'

MQIACF_REGISTRATION_OPTIONS	1081	X'00000439'
MQIACF_PUBLICATION_OPTIONS	1082	X'0000043A'
MQIACF_CLUSTER_INFO	1083	X'0000043B'
MQIACF_Q_MGR_DEFINITION_TYPE	1084	X'0000043C'
MQIACF_Q_MGR_TYPE	1085	X'0000043D'
MQIACF_ACTION	1086	X'0000043E'
MQIACF_SUSPEND	1087	X'0000043F'
MQIACF_BROKER_COUNT	1088	X'00000440'
MQIACF_APPL_COUNT	1089	X'00000441'
MQIACF_ANONYMOUS_COUNT	1090	X'00000442'
MQIACF_REG_REG_OPTIONS	1091	X'00000443'
MQIACF_DELETE_OPTIONS	1092	X'00000444'
MQIACF_CLUSTER_Q_MGR_ATTRS	1093	X'00000445'
MQIACF_REFRESH_INTERVAL	1094	X'00000446'
MQIACF_REFRESH_REPOSITORY	1095	X'00000447'
MQIACF_REMOVE_QUEUES	1096	X'00000448'
MQIACF_OPEN_INPUT_TYPE	1098	X'0000044A'
MQIACF_OPEN_OUTPUT	1099	X'0000044B'
MQIACF_OPEN_SET	1100	X'0000044C'
MQIACF_OPEN_INQUIRE	1101	X'0000044D'
MQIACF_OPEN_BROWSE	1102	X'0000044E'
MQIACF_Q_STATUS_TYPE	1103	X'0000044F'
MQIACF_Q_HANDLE	1104	X'00000450'
MQIACF_Q_STATUS	1105	X'00000451'
MQIACF_SECURITY_TYPE	1106	X'00000452'
MQIACF_CONNECTION_ATTRS	1107	X'00000453'
MQIACF_CONNECT_OPTIONS	1108	X'00000454'
MQIACF_CONN_INFO_TYPE	1110	X'00000456'
MQIACF_CONN_INFO_CONN	1111	X'00000457'
MQIACF_CONN_INFO_HANDLE	1112	X'00000458'
MQIACF_CONN_INFO_ALL	1113	X'00000459'
MQIACF_AUTH_PROFILE_ATTRS	1114	X'0000045A'
MQIACF_AUTHORIZATION_LIST	1115	X'0000045B'
MQIACF_AUTH_ADD_AUTHS	1116	X'0000045C'
MQIACF_AUTH_REMOVE_AUTHS	1117	X'0000045D'
MQIACF_ENTITY_TYPE	1118	X'0000045E'
MQIACF_COMMAND_INFO	1120	X'00000460'
MQIACF_CMDSCOPE_Q_MGR_COUNT	1121	X'00000461'
MQIACF_Q_MGR_SYSTEM	1122	X'00000462'
MQIACF_Q_MGR_EVENT	1123	X'00000463'
MQIACF_Q_MGR_DQM	1124	X'00000464'
MQIACF_Q_MGR_CLUSTER	1125	X'00000465'

MQIACF_QSG_DISPS	1126	X'00000466'
MQIACF_UOW_STATE	1128	X'00000468'
MQIACF_SECURITY_ITEM	1129	X'00000469'
MQIACF_CF_STRUC_STATUS	1130	X'0000046A'
MQIACF_UOW_TYPE	1132	X'0000046C'
MQIACF_CF_STRUC_ATTRS	1133	X'0000046D'
MQIACF_EXCLUDE_INTERVAL	1134	X'0000046E'
MQIACF_CF_STATUS_TYPE	1135	X'0000046F'
MQIACF_CF_STATUS_SUMMARY	1136	X'00000470'
MQIACF_CF_STATUS_CONNECT	1137	X'00000471'
MQIACF_CF_STATUS_BACKUP	1138	X'00000472'
MQIACF_CF_STRUC_TYPE	1139	X'00000473'
MQIACF_CF_STRUC_SIZE_MAX	1140	X'00000474'
MQIACF_CF_STRUC_SIZE_USED	1141	X'00000475'
MQIACF_CF_STRUC_ENTRIES_MAX	1142	X'00000476'
MQIACF_CF_STRUC_ENTRIES_USED	1143	X'00000477'
MQIACF_CF_STRUC_BACKUP_SIZE	1144	X'00000478'
MQIACF_MOVE_TYPE	1145	X'00000479'
MQIACF_MOVE_TYPE_MOVE	1146	X'0000047A'
MQIACF_MOVE_TYPE_ADD	1147	X'0000047B'
MQIACF_Q_MGR_NUMBER	1148	X'0000047C'
MQIACF_Q_MGR_STATUS	1149	X'0000047D'
MQIACF_DB2_CONN_STATUS	1150	X'0000047E'
MQIACF_SECURITY_ATTRS	1151	X'0000047F'
MQIACF_SECURITY_TIMEOUT	1152	X'00000480'
MQIACF_SECURITY_INTERVAL	1153	X'00000481'
MQIACF_SECURITY_SWITCH	1154	X'00000482'
MQIACF_SECURITY_SETTING	1155	X'00000483'
MQIACF_STORAGE_CLASS_ATTRS	1156	X'00000484'
MQIACF_USAGE_TYPE	1157	X'00000485'
MQIACF_BUFFER_POOL_ID	1158	X'00000486'
MQIACF_USAGE_TOTAL_PAGES	1159	X'00000487'
MQIACF_USAGE_UNUSED_PAGES	1160	X'00000488'
MQIACF_USAGE_PERSIST_PAGES	1161	X'00000489'
MQIACF_USAGE_NONPERSIST_PAGES	1162	X'0000048A'
MQIACF_USAGE_RESTART_EXTENTS	1163	X'0000048B'
MQIACF_USAGE_EXPAND_COUNT	1164	X'0000048C'
MQIACF_PAGESET_STATUS	1165	X'0000048D'
MQIACF_USAGE_TOTAL_BUFFERS	1166	X'0000048E'
MQIACF_USAGE_DATA_SET_TYPE	1167	X'0000048F'
MQIACF_USAGE_PAGESET	1168	X'00000490'
MQIACF_USAGE_DATA_SET	1169	X'00000491'

MQIACF_USAGE_BUFFER_POOL	1170	X'00000492'
MQIACF_MOVE_COUNT	1171	X'00000493'
MQIACF_EXPIRY_Q_COUNT	1172	X'00000494'
MQIACF_CONFIGURATION_OBJECTS	1173	X'00000495'
MQIACF_CONFIGURATION_EVENTS	1174	X'00000496'
MQIACF_SYSP_TYPE	1175	X'00000497'
MQIACF_SYSP_DEALLOC_INTERVAL	1176	X'00000498'
MQIACF_SYSP_MAX_ARCHIVE	1177	X'00000499'
MQIACF_SYSP_MAX_READ_TAPES	1178	X'0000049A'
MQIACF_SYSP_IN_BUFFER_SIZE	1179	X'0000049B'
MQIACF_SYSP_OUT_BUFFER_SIZE	1180	X'0000049C'
MQIACF_SYSP_OUT_BUFFER_COUNT	1181	X'0000049D'
MQIACF_SYSP_ARCHIVE	1182	X'0000049E'
MQIACF_SYSP_DUAL_ACTIVE	1183	X'0000049F'
MQIACF_SYSP_DUAL_ARCHIVE	1184	X'000004A0'
MQIACF_SYSP_DUAL_BSDS	1185	X'000004A1'
MQIACF_SYSP_MAX_CONNS	1186	X'000004A2'
MQIACF_SYSP_MAX_CONNS_FORE	1187	X'000004A3'
MQIACF_SYSP_MAX_CONNS_BACK	1188	X'000004A4'
MQIACF_SYSP_EXIT_INTERVAL	1189	X'000004A5'
MQIACF_SYSP_EXIT_TASKS	1190	X'000004A6'
MQIACF_SYSP_CHKPOINT_COUNT	1191	X'000004A7'
MQIACF_SYSP_OTMA_INTERVAL	1192	X'000004A8'
MQIACF_SYSP_Q_INDEX_DEFER	1193	X'000004A9'
MQIACF_SYSP_DB2_TASKS	1194	X'000004AA'
MQIACF_SYSP_RESLEVEL_AUDIT	1195	X'000004AB'
MQIACF_SYSP_ROUTING_CODE	1196	X'000004AC'
MQIACF_SYSP_SMF_ACCOUNTING	1197	X'000004AD'
MQIACF_SYSP_SMF_STATS	1198	X'000004AE'
MQIACF_SYSP_SMF_INTERVAL	1199	X'000004AF'
MQIACF_SYSP_TRACE_CLASS	1200	X'000004B0'
MQIACF_SYSP_TRACE_SIZE	1201	X'000004B1'
MQIACF_SYSP_WLM_INTERVAL	1202	X'000004B2'
MQIACF_SYSP_ALLOC_UNIT	1203	X'000004B3'
MQIACF_SYSP_ARCHIVE_RETAIN	1204	X'000004B4'
MQIACF_SYSP_ARCHIVE_WTOR	1205	X'000004B5'
MQIACF_SYSP_BLOCK_SIZE	1206	X'000004B6'
MQIACF_SYSP_CATALOG	1207	X'000004B7'
MQIACF_SYSP_COMPACT	1208	X'000004B8'
MQIACF_SYSP_ALLOC_PRIMARY	1209	X'000004B9'
MQIACF_SYSP_ALLOC_SECONDARY	1210	X'000004BA'
MQIACF_SYSP_PROTECT	1211	X'000004BB'

MQIACF_SYSP_QUIESCE_INTERVAL	1212	X'000004BC'
MQIACF_SYSP_TIMESTAMP	1213	X'000004BD'
MQIACF_SYSP_UNIT_ADDRESS	1214	X'000004BE'
MQIACF_SYSP_UNIT_STATUS	1215	X'000004BF'
MQIACF_SYSP_LOG_COPY	1216	X'000004C0'
MQIACF_SYSP_LOG_USED	1217	X'000004C1'
MQIACF_SYSP_LOG_SUSPEND	1218	X'000004C2'
MQIACF_SYSP_OFFLOAD_STATUS	1219	X'000004C3'
MQIACF_SYSP_TOTAL_LOGS	1220	X'000004C4'
MQIACF_SYSP_FULL_LOGS	1221	X'000004C5'
MQIACF_LISTENER_ATTRS	1222	X'000004C6'
MQIACF_LISTENER_STATUS_ATTRS	1223	X'000004C7'
MQIACF_SERVICE_ATTRS	1224	X'000004C8'
MQIACF_SERVICE_STATUS_ATTRS	1225	X'000004C9'
MQIACF_Q_TIME_INDICATOR	1226	X'000004CA'
MQIACF_OLDEST_MSG_AGE	1227	X'000004CB'
MQIACF_AUTH_OPTIONS	1228	X'000004CC'
MQIACF_Q_MGR_STATUS_ATTRS	1229	X'000004CD'
MQIACF_CONNECTION_COUNT	1230	X'000004CE'
MQIACF_Q_MGR_FACILITY	1231	X'000004CF'
MQIACF_CHINIT_STATUS	1232	X'000004D0'
MQIACF_CMD_SERVER_STATUS	1233	X'000004D1'
MQIACF_ROUTE_DETAIL	1234	X'000004D2'
MQIACF_RECORDED_ACTIVITIES	1235	X'000004D3'
MQIACF_MAX_ACTIVITIES	1236	X'000004D4'
MQIACF_DISCONTINUITY_COUNT	1237	X'000004D5'
MQIACF_ROUTE_ACCUMULATION	1238	X'000004D6'
MQIACF_ROUTE_DELIVERY	1239	X'000004D7'
MQIACF_OPERATION_TYPE	1240	X'000004D8'
MQIACF_BACKOUT_COUNT	1241	X'000004D9'
MQIACF_COMP_CODE	1242	X'000004DA'
MQIACF_ENCODING	1243	X'000004DB'
MQIACF_EXPIRY	1244	X'000004DC'
MQIACF_FEEDBACK	1245	X'000004DD'
MQIACF_MSG_FLAGS	1247	X'000004DF'
MQIACF_MSG_LENGTH	1248	X'000004E0'
MQIACF_MSG_TYPE	1249	X'000004E1'
MQIACF_OFFSET	1250	X'000004E2'
MQIACF_ORIGINAL_LENGTH	1251	X'000004E3'
MQIACF_PERSISTENCE	1252	X'000004E4'
MQIACF_PRIORITY	1253	X'000004E5'
MQIACF_REASON_CODE	1254	X'000004E6'

MQIACF_REPORT	1255	X'000004E7'
MQIACF_VERSION	1256	X'000004E8'
MQIACF_UNRECORDED_ACTIVITIES	1257	X'000004E9'
MQIACF_MONITORING	1258	X'000004EA'
MQIACF_ROUTE_FORWARDING	1259	X'000004EB'
MQIACF_SERVICE_STATUS	1260	X'000004EC'
MQIACF_Q_TYPES	1261	X'000004ED'
MQIACF_USER_ID_SUPPORT	1262	X'000004EE'
MQIACF_INTERFACE_VERSION	1263	X'000004EF'
MQIACF_AUTH_SERVICE_ATTRS	1264	X'000004F0'
MQIACF_USAGE_EXPAND_TYPE	1265	X'000004F1'
MQIACF_SYSP_CLUSTER_CACHE	1266	X'000004F2'
MQIACF_SYSP_DB2_BLOB_TASKS	1267	X'000004F3'
MQIACF_SYSP_WLM_INT_UNITS	1268	X'000004F4'
MQIACF_TOPIC_ATTRS	1269	X'000004F5'
MQIACF_PUBSUB_PROPERTIES	1271	X'000004F7'
MQIACF_DESTINATION_CLASS	1273	X'000004F9'
MQIACF_DURABLE_SUBSCRIPTION	1274	X'000004FA'
MQIACF_SUBSCRIPTION_SCOPE	1275	X'000004FB'
MQIACF_VARIABLE_USER_ID	1277	X'000004FD'
MQIACF_REQUEST_ONLY	1280	X'00000500'
MQIACF_PUB_PRIORITY	1283	X'00000503'
MQIACF_SUB_ATTRS	1287	X'00000507'
MQIACF_WILDCARD_SCHEMA	1288	X'00000508'
MQIACF_SUB_TYPE	1289	X'00000509'
MQIACF_MESSAGE_COUNT	1290	X'0000050A'
MQIACF_Q_MGR_PUBSUB	1291	X'0000050B'
MQIACF_Q_MGR_VERSION	1292	X'0000050C'
MQIACF_SUB_STATUS_ATTRS	1294	X'0000050E'
MQIACF_TOPIC_STATUS	1295	X'0000050F'
MQIACF_TOPIC_SUB	1296	X'00000510'
MQIACF_TOPIC_PUB	1297	X'00000511'
MQIACF_RETAINED_PUBLICATION	1300	X'00000514'
MQIACF_TOPIC_STATUS_ATTRS	1301	X'00000515'
MQIACF_TOPIC_STATUS_TYPE	1302	X'00000516'
MQIACF_SUB_OPTIONS	1303	X'00000517'
MQIACF_PUBLISH_COUNT	1304	X'00000518'
MQIACF_CLEAR_TYPE	1305	X'00000519'
MQIACF_CLEAR_SCOPE	1306	X'0000051A'
MQIACF_SUB_LEVEL	1307	X'0000051B'
MQIACF_ASYNC_STATE	1308	X'0000051C'
MQIACF_SUB_SUMMARY	1309	X'0000051D'

MQIACF_OBSOLETE_MSGS	1310	X'0000051E'
MQIACF_PUBSUB_STATUS	1311	X'0000051F'
MQIACF_PS_STATUS_TYPE	1314	X'00000522'
MQIACF_PUBSUB_STATUS_ATTRS	1318	X'00000526'
MQIACF_SELECTOR_TYPE	1321	X'00000529'
MQIACF_MCAST_REL_INDICATOR	1351	X'00000547'
MQIACF_LAST_USED	1351	X'00000547'

MQIACH_* (Command format Integer Channel Types):

MQIACH_FIRST		1501	X'000005DD'
MQIACH_XMIT_PROTOCOL_TYPE		1501	X'000005DD'
MQIACH_BATCH_SIZE		1502	X'000005DE'
MQIACH_DISC_INTERVAL		1503	X'000005DF'
MQIACH_SHORT_TIMER		1504	X'000005E0'
MQIACH_SHORT_RETRY		1505	X'000005E1'
MQIACH_LONG_TIMER		1506	X'000005E2'
MQIACH_LONG_RETRY		1507	X'000005E3'
MQIACH_PUT_AUTHORITY		1508	X'000005E4'
MQIACH_SEQUENCE_NUMBER_WRAP		1509	X'000005E5'
MQIACH_MAX_MSG_LENGTH		1510	X'000005E6'
MQIACH_CHANNEL_TYPE		1511	X'000005E7'
MQIACH_DATA_COUNT		1512	X'000005E8'
MQIACH_NAME_COUNT		1513	X'000005E9'
MQIACH_MSG_SEQUENCE_NUMBER		1514	X'000005EA'
MQIACH_DATA_CONVERSION		1515	X'000005EB'
MQIACH_IN_DOUBT		1516	X'000005EC'
MQIACH_MCA_TYPE		1517	X'000005ED'
MQIACH_SESSION_COUNT		1518	X'000005EE'
MQIACH_ADAPTER		1519	X'000005EF'
MQIACH_COMMAND_COUNT		1520	X'000005F0'
MQIACH_SOCKET		1521	X'000005F1'
MQIACH_PORT		1522	X'000005F2'
MQIACH_CHANNEL_INSTANCE_TYPE		1523	X'000005F3'
MQIACH_CHANNEL_INSTANCE_ATTRS		1524	X'000005F4'
MQIACH_CHANNEL_ERROR_DATA		1525	X'000005F5'
MQIACH_CHANNEL_TABLE		1526	X'000005F6'
MQIACH_CHANNEL_STATUS		1527	X'000005F7'
MQIACH_INDOUBT_STATUS		1528	X'000005F8'
MQIACH_LAST_SEQ_NUMBER		1529	X'000005F9'
MQIACH_LAST_SEQUENCE_NUMBER		1529	X'000005F9'
MQIACH_CURRENT_MSGS		1531	X'000005FB'

MQIACH_CURRENT_SEQ_NUMBER		1532	X'000005FC'
MQIACH_CURRENT_SEQUENCE_NUMBER		1532	X'000005FC'
MQIACH_SSL_RETURN_CODE		1533	X'000005FD'
MQIACH_MSGS		1534	X'000005FE'
MQIACH_BYTES_SENT		1535	X'000005FF'
MQIACH_BYTES_RCVD		1536	X'00000600'
MQIACH_BYTES_RECEIVED		1536	X'00000600'
MQIACH_BATCHES		1537	X'00000601'
MQIACH_BUFFERS_SENT		1538	X'00000602'
MQIACH_BUFFERS_RCVD		1539	X'00000603'
MQIACH_BUFFERS_RECEIVED		1539	X'00000603'
MQIACH_LONG_RETRIES_LEFT		1540	X'00000604'
MQIACH_SHORT_RETRIES_LEFT		1541	X'00000605'
MQIACH_MCA_STATUS		1542	X'00000606'
MQIACH_STOP_REQUESTED		1543	X'00000607'
MQIACH_MR_COUNT		1544	X'00000608'
MQIACH_MR_INTERVAL		1545	X'00000609'
These rows intentionally left blank			
MQIACH_NPM_SPEED		1562	X'0000061A'
MQIACH_HB_INTERVAL		1563	X'0000061B'
MQIACH_BATCH_INTERVAL		1564	X'0000061C'
MQIACH_NETWORK_PRIORITY		1565	X'0000061D'
MQIACH_KEEP_ALIVE_INTERVAL		1566	X'0000061E'
MQIACH_BATCH_HB		1567	X'0000061F'
MQIACH_SSL_CLIENT_AUTH		1568	X'00000620'
This row intentionally left blank			
MQIACH_ALLOC_RETRY		1570	X'00000622'
MQIACH_ALLOC_FAST_TIMER		1571	X'00000623'
MQIACH_ALLOC_SLOW_TIMER		1572	X'00000624'
MQIACH_DISC_RETRY		1573	X'00000625'
MQIACH_PORT_NUMBER		1574	X'00000626'
MQIACH_HDR_COMPRESSION		1575	X'00000627'
MQIACH_MSG_COMPRESSION		1576	X'00000628'
MQIACH_CLWL_CHANNEL_RANK		1577	X'00000629'
MQIACH_CLWL_CHANNEL_PRIORITY		1578	X'0000062A'
MQIACH_CLWL_CHANNEL_WEIGHT		1579	X'0000062B'
MQIACH_CHANNEL_DISP		1580	X'0000062C'
MQIACH_INBOUND_DISP		1581	X'0000062D'
MQIACH_CHANNEL_TYPES		1582	X'0000062E'
MQIACH_ADAPS_STARTED		1583	X'0000062F'
MQIACH_ADAPS_MAX		1584	X'00000630'
MQIACH_DISPS_STARTED		1585	X'00000631'

MQIACH_DISPS_MAX		1586	X'00000632'
MQIACH_SSLTASKS_STARTED		1587	X'00000633'
MQIACH_SSLTASKS_MAX		1588	X'00000634'
MQIACH_CURRENT_CHL		1589	X'00000635'
MQIACH_CURRENT_CHL_MAX		1590	X'00000636'
MQIACH_CURRENT_CHL_TCP		1591	X'00000637'
MQIACH_CURRENT_CHL_LU62		1592	X'00000638'
MQIACH_ACTIVE_CHL		1593	X'00000639'
MQIACH_ACTIVE_CHL_MAX		1594	X'0000063A'
MQIACH_ACTIVE_CHL_PAUSED		1595	X'0000063B'
MQIACH_ACTIVE_CHL_STARTED		1596	X'0000063C'
MQIACH_ACTIVE_CHL_STOPPED		1597	X'0000063D'
MQIACH_ACTIVE_CHL_RETRY		1598	X'0000063E'
MQIACH_LISTENER_STATUS		1599	X'0000063F'
MQIACH_SHARED_CHL_RESTART		1600	X'00000640'
MQIACH_LISTENER_CONTROL		1601	X'00000641'
MQIACH_BACKLOG		1602	X'00000642'
MQIACH_XMITQ_TIME_INDICATOR		1604	X'00000644'
MQIACH_NETWORK_TIME_INDICATOR		1605	X'00000645'
MQIACH_EXIT_TIME_INDICATOR		1606	X'00000646'
MQIACH_BATCH_SIZE_INDICATOR		1607	X'00000647'
MQIACH_XMITQ_MSGS_AVAILABLE		1608	X'00000648'
MQIACH_CHANNEL_SUBSTATE		1609	X'00000649'
MQIACH_SSL_KEY_RESETS		1610	X'0000064A'
MQIACH_COMPRESSION_RATE		1611	X'0000064B'
MQIACH_COMPRESSION_TIME		1612	X'0000064C'
MQIACH_MAX_XMIT_SIZE		1613	X'0000064D'
MQIACH_DEF_CHANNEL_DISP		1614	X'0000064E'
MQIACH_SHARING_CONVERSATIONS		1615	X'0000064F'
MQIACH_MAX_SHARING_CONVS		1616	X'00000650'
MQIACH_CURRENT_SHARING_CONVS		1617	X'00000651'
MQIACH_MAX_INSTANCES		1618	X'00000652'
MQIACH_MAX_INSTS_PER_CLIENT		1619	X'00000653'
MQIACH_CLIENT_CHANNEL_WEIGHT		1620	X'00000654'
MQIACH_CONNECTION_AFFINITY		1621	X'00000655'
This row intentionally left blank			
MQIACH_RESET_REQUESTED		1623	X'00000657'
MQIACH_BATCH_DATA_LIMIT		1624	X'00000658'
MQIACH_MSG_HISTORY		1625	X'00000659'
MQIACH_MULTICAST_PROPERTIES		1626	X'0000065A'
MQIACH_NEW_SUBSCRIBER_HISTORY		1627	X'0000065B'
MQIACH_MC_HB_INTERVAL		1628	X'0000065C'

MQIACH_USE_CLIENT_ID		1629	X'0000065D'
MQIACH_MQTT_KEEP_ALIVE		1630	X'0000065E'
MQIACH_IN_DOUBT_IN		1631	X'0000065F'
MQIACH_IN_DOUBT_OUT		1632	X'00000660'
MQIACH_MSGS_SENT<		1633	X'00000661'
MQIACH_MSGS_RECEIVED		1634	X'00000662'
MQIACH_MSGS_RCVD		1634	X'00000662'
MQIACH_PENDING_OUT		1635	X'00000663'
MQIACH_AVAILABLE_CIPHERSPECS		1636	X'00000664'
MQIACH_MATCH		1637	X'00000665'
MQIACH_USER_SOURCE		1638	X'00000666'
MQIACH_WARNING		1639	X'00000667'
MQIACH_DEF_RECONNECT		1640	X'00000668'
This row intentionally left blank			
MQIACH_CHANNEL_SUMMARY_ATTRS		1642	X'0000066A'
MQIACH_LAST_USED		1642	X'0000066A'

MQIAMO_* (Command format Integer Monitoring Parameter Types):

MQIAMO_FIRST	701	X'000002BD'
MQIAMO_AVG_BATCH_SIZE	702	X'000002BE'
MQIAMO_AVG_Q_TIME	703	X'000002BF'
MQIAMO_BACKOUTS	704	X'000002C0'
MQIAMO_BROWSES	705	X'000002C1'
MQIAMO_BROWSE_MAX_BYTES	706	X'000002C2'
MQIAMO_BROWSE_MIN_BYTES	707	X'000002C3'
MQIAMO_BROWSES_FAILED	708	X'000002C4'
MQIAMO_CLOSES	709	X'000002C5'
MQIAMO_COMMITS	710	X'000002C6'
MQIAMO_COMMITS_FAILED	711	X'000002C7'
MQIAMO_CONNS	712	X'000002C8'
MQIAMO_CONNS_MAX	713	X'000002C9'
MQIAMO_DISCS	714	X'000002CA'
MQIAMO_DISCS_IMPLICIT	715	X'000002CB'
MQIAMO_DISC_TYPE	716	X'000002CC'
MQIAMO_EXIT_TIME_AVG	717	X'000002CD'
MQIAMO_EXIT_TIME_MAX	718	X'000002CE'
MQIAMO_EXIT_TIME_MIN	719	X'000002CF'
MQIAMO_FULL_BATCHES	720	X'000002D0'
MQIAMO_GENERATED_MSGS	721	X'000002D1'
MQIAMO_GETS	722	X'000002D2'
MQIAMO_GET_MAX_BYTES	723	X'000002D3'

MQIAMO_GET_MIN_BYTES	724	X'000002D4'
MQIAMO_GETS_FAILED	725	X'000002D5'
MQIAMO_INCOMPLETE_BATCHES	726	X'000002D6'
MQIAMO_INQS	727	X'000002D7'
MQIAMO_MSGS	728	X'000002D8'
MQIAMO_NET_TIME_AVG	729	X'000002D9'
MQIAMO_NET_TIME_MAX	730	X'000002DA'
MQIAMO_NET_TIME_MIN	731	X'000002DB'
MQIAMO_OBJECT_COUNT	732	X'000002DC'
MQIAMO_OPENS	733	X'000002DD'
MQIAMO_PUT1S	734	X'000002DE'
MQIAMO_PUTS	735	X'000002DF'
MQIAMO_PUT_MAX_BYTES	736	X'000002E0'
MQIAMO_PUT_MIN_BYTES	737	X'000002E1'
MQIAMO_PUT_RETRIES	738	X'000002E2'
MQIAMO_Q_MAX_DEPTH	739	X'000002E3'
MQIAMO_Q_MIN_DEPTH	740	X'000002E4'
MQIAMO_Q_TIME_AVG	741	X'000002E5'
MQIAMO_Q_TIME_MAX	742	X'000002E6'
MQIAMO_Q_TIME_MIN	743	X'000002E7'
MQIAMO_SETS	744	X'000002E8'
MQIAMO_CONNS_FAILED	749	X'000002ED'
MQIAMO_OPENS_FAILED	751	X'000002EF'
MQIAMO_INQS_FAILED	752	X'000002F0'
MQIAMO_SETS_FAILED	753	X'000002F1'
MQIAMO_PUTS_FAILED	754	X'000002F2'
MQIAMO_PUT1S_FAILED	755	X'000002F3'
MQIAMO_CLOSES_FAILED	757	X'000002F5'
MQIAMO_MSGS_EXPIRED	758	X'000002F6'
MQIAMO_MSGS_NOT_QUEUED	759	X'000002F7'
MQIAMO_MSGS_PURGED	760	X'000002F8'
MQIAMO_SUBS_DUR	764	X'000002FC'
MQIAMO_SUBS_NDUR	765	X'000002FD'
MQIAMO_SUBS_FAILED	766	X'000002FE'
MQIAMO_SUBRQS	767	X'000002FF'
MQIAMO_SUBRQS_FAILED	768	X'00000300'
MQIAMO_CBS	769	X'00000301'
MQIAMO_CBS_FAILED	770	X'00000302'
MQIAMO_CTLS	771	X'00000303'
MQIAMO_CTLS_FAILED	772	X'00000304'
MQIAMO_STATS	773	X'00000305'
MQIAMO_STATS_FAILED	774	X'00000306'

MQIAMO_SUB_DUR_HIGHWATER	775	X'00000307'
MQIAMO_SUB_DUR_LOWWATER	776	X'00000308'
MQIAMO_SUB_NDUR_HIGHWATER	777	X'00000309'
MQIAMO_SUB_NDUR_LOWWATER	778	X'0000030A'
MQIAMO_TOPIC_PUTS	779	X'0000030B'
MQIAMO_TOPIC_PUTS_FAILED	780	X'0000030C'
MQIAMO_TOPIC_PUT1S	781	X'0000030D'
MQIAMO_TOPIC_PUT1S_FAILED	782	X'0000030E'
MQIAMO_PUBLISH_MSG_COUNT	784	X'00000310'
MQIAMO_UNSUBS_DUR	786	X'00000312'
MQIAMO_UNSUBS_NDUR	787	X'00000313'
MQIAMO_UNSUBS_FAILED	788	X'00000314'
MQIAMO_INTERVAL	789	X'00000315'
MQIAMO_MSGS_SENT	790	X'00000316'
MQIAMO_BYTES_SENT	791	X'00000317'
MQIAMO_REPAIR_BYTES	792	X'00000318'
MQIAMO_FEEDBACK_MODE	793	X'00000319'
MQIAMO_RELIABILITY_TYPE	794	X'0000031A'
MQIAMO_LATE_JOIN_MARK	795	X'0000031B'
MQIAMO_NACKS_RCVD	796	X'0000031C'
MQIAMO_REPAIR_PKTS	797	X'0000031D'
MQIAMO_HISTORY_PKTS	798	X'0000031E'
MQIAMO_PENDING_PKTS	799	X'0000031F'
MQIAMO_PKT_RATE	800	X'00000320'
MQIAMO_MCAST_XMIT_RATE	801	X'00000321'
MQIAMO_MCAST_BATCH_TIME	802	X'00000322'
MQIAMO_MCAST_HEARTBEAT	803	X'00000323'
MQIAMO_DEST_DATA_PORT	804	X'00000324'
MQIAMO_DEST_REPAIR_PORT	805	X'00000325'
MQIAMO_ACKS_RCVD	806	X'00000326'
MQIAMO_ACTIVE_ACKERS	807	X'00000327'
MQIAMO_PKTS_SENT	808	X'00000328'
MQIAMO_TOTAL_REPAIR_PKTS	809	X'00000329'
MQIAMO_TOTAL_PKTS_SENT	810	X'0000032A'
MQIAMO_TOTAL_MSGS_SENT	811	X'0000032B'
MQIAMO_TOTAL_BYTES_SENT	812	X'0000032C'
MQIAMO_NUM_STREAMS	813	X'0000032D'
MQIAMO_ACK_FEEDBACK	814	X'0000032E'
MQIAMO_NACK_FEEDBACK	815	X'0000032F'
MQIAMO_PKTS_LOST	816	X'00000330'
MQIAMO_MSGS_RCVD	817	X'00000331'
MQIAMO_MSG_BYTES_RCVD	818	X'00000332'

MQIAMO_MSGS_DELIVERED	819	X'00000333'
MQIAMO_PKTS_PROCESSED	820	X'00000334'
MQIAMO_PKTS_DLVD	821	X'00000335'
MQIAMO_PKTS_DROPPED	822	X'00000336'
MQIAMO_PKTS_DUPLICATED	823	X'00000337'
MQIAMO_NACKS_CREATED	824	X'00000338'
MQIAMO_NACK_PKTS_SENT	825	X'00000339'
MQIAMO_REPAIR_PKTS_RQSTD	826	X'0000033A'
MQIAMO_REPAIR_PKTS_RCVD	827	X'0000033B'
MQIAMO_PKTS_REPAIRED	828	X'0000033C'
MQIAMO_TOTAL_MSGS_RCVD	829	X'0000033D'
MQIAMO_TOTAL_MSGS_BYTES_RCVD	830	X'0000033E'
MQIAMO_TOTAL_REPAIR_PKTS_RCVD	831	X'0000033F'
MQIAMO_TOTAL_REPAIR_PKTS_RQSTD	832	X'00000340'
MQIAMO_TOTAL_MSGS_PROCESSED	833	X'00000341'
MQIAMO_TOTAL_MSGS_SELECTED	834	X'00000342'
MQIAMO_TOTAL_MSGS_EXPIRED	835	X'00000343'
MQIAMO_TOTAL_MSGS_DELIVERED	836	X'00000344'
MQIAMO_TOTAL_MSGS_RETURNED	837	X'00000345'
MQIAMO_LAST_USED	837	X'00000345'

MQIAMO64_* (Command format 64-bit Integer Monitoring Parameter Types):

MQIAMO64_AVG_Q_TIME	703	X'000002BF'
MQIAMO64_Q_TIME_AVG	741	X'000002E5'
MQIAMO64_Q_TIME_MAX	742	X'000002E6'
MQIAMO64_Q_TIME_MIN	743	X'000002E7'
MQIAMO64_BROWSE_BYTES	745	X'000002E9'
MQIAMO64_BYTES	746	X'000002EA'
MQIAMO64_GET_BYTES	747	X'000002EB'
MQIAMO64_PUT_BYTES	748	X'000002EC'
MQIAMO64_TOPIC_PUT_BYTES	783	X'0000030F'
MQIAMO64_PUBLISH_MSG_BYTES	785	X'00000311'

MQIASY_* (Integer System Selectors):

MQIASY_FIRST	-1	X'FFFFFFFF'
MQIASY_CODED_CHAR_SET_ID	-1	X'FFFFFFFF'
MQIASY_TYPE	-2	X'FFFFFFFE'
MQIASY_COMMAND	-3	X'FFFFFFFD'
MQIASY_MSG_SEQ_NUMBER	-4	X'FFFFFFFC'
MQIASY_CONTROL	-5	X'FFFFFFFB'
MQIASY_COMP_CODE	-6	X'FFFFFFFA'
MQIASY_REASON	-7	X'FFFFFFF9'
MQIASY_BAG_OPTIONS	-8	X'FFFFFFF8'
MQIASY_VERSION	-9	X'FFFFFFF7'
MQIASY_LAST_USED	-9	X'FFFFFFF7'
MQIASY_LAST	-2000	X'FFFFF830'

MQIAUT_* (IMS information header Authenticator):

MQIAUT_NONE	"bbbbbbbb"
MQIAUT_NONE_ARRAY	'b','b','b','b','b','b','b','b'

MQIAV_* (Integer Attribute Values):

MQIAV_NOT_APPLICABLE	-1	X'FFFFFFFF'
MQIAV_UNDEFINED	-2	X'FFFFFFFE'

MQICM_* (IMS information header Commit Modes):

MQICM_COMMIT_THEN_SEND	'0'	
MQICM_SEND_THEN_COMMIT	'1'	

MQIDO_* (Command format Indoubt Options):

MQIDO_COMMIT	1	X'00000001'
MQIDO_BACKOUT	2	X'00000002'

MQIEP_* (Interface entry points):

MQIEP_STRUC_ID	"IEP "	
MQIEP_STRUC_ID_ARRAY	'I','E','P',' '	
MQIEP_VERSION_1	1	X'00000001'
MQDXP_CURRENT_VERSION	1	X'00000001'

MQIGQ_* (Intra-Group Queuing):

MQIGQ_DISABLED	0	X'00000000'
MQIGQ_ENABLED	1	X'00000001'

MQIGQPA_* (Intra-Group Queuing Put Authority):

MQIGQPA_DEFAULT	1	X'00000001'
MQIGQPA_CONTEXT	2	X'00000002'
MQIGQPA_ONLY_IGQ	3	X'00000003'
MQIGQPA_ALTERNATE_OR_IGQ	4	X'00000004'

MQIIH_* (IMS information header structure and Flags):
IMS information header structure

MQIIH_STRUC_ID	"IIHb"	
MQIIH_STRUC_ID_ARRAY	'I','I','H','b'	
MQIIH_VERSION_1	1	X'00000001'
MQIIH_CURRENT_VERSION	1	X'00000001'
MQIIH_LENGTH_1	84	X'00000054'

IMS information header Flags

MQIIH_NONE	0	X'00000000'
MQIIH_PASS_EXPIRATION	1	X'00000001'
MQIIH_UNLIMITED_EXPIRATION	0	X'00000000'
MQIIH_REPLY_FORMAT_NONE	8	X'00000008'
MQIIH_IGNORE_PURG	16	X'00000010'
MQIIH_CM0_REQUEST_RESPONSE	32	X'00000020'

MQIMPO_* (Inquire message property options and structure):
Inquire message property options structure

MQIMPO_STRUC_ID	"IMPO"	
MQIMPO_STRUC_ID_ARRAY	'I','M','P','O'	
MQIMPO_VERSION_1	1	X'00000001'
MQIMPO_CURRENT_VERSION	1	X'00000001'

Inquire Message Property Options

MQIMPO_CONVERT_TYPE	2	X'00000002'
MQIMPO_QUERY_LENGTH	4	X'00000004'
MQIMPO_INQ_FIRST	0	X'00000000'
MQIMPO_INQ_NEXT	8	X'00000008'
MQIMPO_INQ_PROP_UNDER_CURSOR	16	X'00000010'
MQIMPO_CONVERT_VALUE	32	X'00000020'
MQIMPO_NONE	0	X'00000000'

MQINBD_* (Command format Inbound Dispositions):

MQINBD_Q_MGR	0	X'00000000'
MQINBD_GROUP	3	X'00000003'

MQIND_* (Special Index Values):

MQIND_NONE	-1	X'FFFFFFFF'
MQIND_ALL	-2	X'FFFFFFFE'

MQIPADDR_* (IP Address Versions):

MQIPADDR_IPV4	0	X'00000000'
MQIPADDR_IPV6	1	X'00000001'

MQISS_* (IMS information header Security Scopes):

MQISS_CHECK	'C'	
MQISS_FULL	'F'	

MQIT_* (Index Types):

MQIT_NONE	0	X'00000000'
MQIT_MSG_ID	1	X'00000001'
MQIT_CORREL_ID	2	X'00000002'
MQIT_MSG_TOKEN	4	X'00000004'
MQIT_GROUP_ID	5	X'00000005'

MQITEM_* (Item Type for mqInquireItemInfo):

MQITEM_INTEGER	1	X'00000001'
MQITEM_STRING	2	X'00000002'
MQITEM_BAG	3	X'00000003'
MQITEM_BYTE_STRING	4	X'00000004'
MQITEM_INTEGER_FILTER	5	X'00000005'
MQITEM_STRING_FILTER	6	X'00000006'
MQITEM_INTEGER64	7	X'00000007'
MQITEM_BYTE_STRING_FILTER	8	X'00000008'

MQITII_* (IMS information header Transaction Instance Identifier):

MQITII_NONE	X'00...00'	(16 nulls)
MQITII_NONE_ARRAY	'\0','\0',...	(16 nulls)

MQITS_* (IMS information header Transaction States):

MQITS_IN_CONVERSATION	'C'	
MQITS_NOT_IN_CONVERSATION	'b'	
MQITS_ARCHITECTED	'A'	

MQKAI_* (KeepAlive Interval):

MQKAI_AUTO	-1	X'FFFFFFFF'

MQMASTER_* (Master administration):

MQMASTER_NO	0	X'00000000'
MQMASTER_YES	1	X'00000001'

MQMCAS_* (Command format Message Channel Agent Status):

MQMCAS_STOPPED	0	X'00000000'
MQMCAS_RUNNING	3	X'00000003'

MQMCAT_* (MCA Types):

MQMCAT_PROCESS	1	X'00000001'
MQMCAT_THREAD	2	X'00000002'

MQMCD_* (Publish/Subscribe Options Tag Information):
Publish/Subscribe Options Tag Message Content Descriptor (mcd) Tags

MQMCD_FOLDER_VERSION	1	X'00000001'

Publish/Subscribe Options Tag Tag names

MQMCD_MSG_DOMAIN	"Msd"	
MQMCD_MSG_SET	"Set"	
MQMCD_MSG_TYPE	"Type"	
MQMCD_MSG_FORMAT	"Fmt"	

Publish/Subscribe Options Tag XML tag names

MQMCD_MSG_DOMAIN_B	"<Msd>"	
MQMCD_MSG_DOMAIN_E	"</Msd>"	
MQMCD_MSG_SET_B	"<Set>"	
MQMCD_MSG_SET_E	"</Set>"	
MQMCD_MSG_TYPE_B	"<Type>"	
MQMCD_MSG_TYPE_E	"</Type>"	
MQMCD_MSG_FORMAT_B	"<Fmt>"	
MQMCD_MSG_FORMAT_E	"</Fmt>"	

Publish/Subscribe Options Tag Tag values

MQMCD_DOMAIN_NONE	"none"	
MQMCD_DOMAIN_NEON	"neon"	
MQMCD_DOMAIN_MRM	"mrm"	
MQMCD_DOMAIN_JMS_NONE	"jms_none"	
MQMCD_DOMAIN_JMS_TEXT	"jms_text"	
MQMCD_DOMAIN_JMS_OBJECT	"jms_object"	
MQMCD_DOMAIN_JMS_MAP	"jms_map"	
MQMCD_DOMAIN_JMS_STREAM	"jms_stream"	
MQMCD_DOMAIN_JMS_BYTES	"jms_bytes"	

MQMD_* (Message descriptor structure):

MQMD_STRUC_ID	"MDbb"	
MQMD_STRUC_ID_ARRAY	'M','D','b','b'	
MQMD_VERSION_1	1	X'00000001'
MQMD_VERSION_2	2	X'00000002'
MQMD_CURRENT_VERSION	2	X'00000002'

MQMDE_* (Message descriptor extension structure):

MQMDE_STRUC_ID	"MDEb"	
MQMDE_STRUC_ID_ARRAY	'M','D','E','b'	
MQMDE_VERSION_2	2	X'00000002'
MQMDE_CURRENT_VERSION	2	X'00000002'
MQMDE_LENGTH_2	72	X'00000048'

MQMDEF_* (Message descriptor extension Flags):

MQMDEF_NONE	0	X'00000000'

MQMDS_* (Message Delivery Sequence):

MQMDS_PRIORITY	0	X'00000000'
MQMDS_FIFO	1	X'00000001'

MQMF_* (Message Flags):

MQMF_SEGMENTATION_INHIBITED	0	X'00000000'
MQMF_SEGMENTATION_ALLOWED	1	X'00000001'
MQMF_MSG_IN_GROUP	8	X'00000008'
MQMF_LAST_MSG_IN_GROUP	16	X'00000010'
MQMF_SEGMENT	2	X'00000002'
MQMF_LAST_SEGMENT	4	X'00000004'
MQMF_NONE	0	X'00000000'

MQMHBO_* (Message handle to buffer options and structure):
Message handle to buffer options structure

MQMHBO_STRUC_ID	"MHBO"	
MQMHBO_STRUC_ID_ARRAY	'M','H','B','O'	
MQMHBO_VERSION_1	1	X'00000001'
MQMHBO_CURRENT_VERSION	1	X'00000001'

Message Handle To Buffer Options

MQMHBO_PROPERTIES_IN_MQRFH2	1	X'00000001'
MQMHBO_DELETE_PROPERTIES	2	X'00000002'
MQMHBO_NONE	0	X'00000000'

MQMI_* (Message Identifier):

MQMI_NONE	X'00...00'	(24 nulls)
MQMI_NONE_ARRAY	'\0','\0',...	(24 nulls)

MQMMBI_* (Message Mark-Browse Interval):

MQMMBI_UNLIMITED	-1	X'FFFFFFFF'

MQMO_* (Match Options):

MQMO_MATCH_MSG_ID	1	X'00000001'
MQMO_MATCH_CORREL_ID	2	X'00000002'
MQMO_MATCH_GROUP_ID	4	X'00000004'
MQMO_MATCH_MSG_SEQ_NUMBER	8	X'00000008'
MQMO_MATCH_OFFSET	16	X'00000010'
MQMO_MATCH_MSG_TOKEN	32	X'00000020'
MQMO_NONE	0	X'00000000'

MQMODE_* (Command format Mode Options):

MQMODE_FORCE	0	X'00000000'
MQMODE_QUIESCE	1	X'00000001'
MQMODE_TERMINATE	2	X'00000002'

MQMON_* (Monitoring Values):

MQMON_NOT_AVAILABLE	-1	X'FFFFFFFF'
MQMON_NONE	-1	X'FFFFFFFF'
MQMON_Q_MGR	-3	X'FFFFFFFD'
MQMON_OFF	0	X'00000000'
MQMON_ON	1	X'00000001'
MQMON_DISABLED	0	X'00000000'
MQMON_ENABLED	1	X'00000001'
MQMON_LOW	17	X'00000011'
MQMON_MEDIUM	33	X'00000021'
MQMON_HIGH	65	X'00000041'

MQMT_* (Message Types):

MQMT_SYSTEM_FIRST	1	X'00000001'
MQMT_REQUEST	1	X'00000001'
MQMT_REPLY	2	X'00000002'
MQMT_DATAGRAM	8	X'00000008'
MQMT_REPORT	4	X'00000004'
MQMT_MQE_FIELDS_FROM_MQE	112	X'00000070'
MQMT_MQE_FIELDS	113	X'00000071'
MQMT_SYSTEM_LAST	65535	X'0000FFFF'
MQMT_APPL_FIRST	65536	X'00010000'
MQMT_APPL_LAST	999999999	X'3B9AC9FF'

MQMTOK_* (Message Token):

MQMTOK_NONE	X'00...00'	(16 nulls)
MQMTOK_NONE_ARRAY	'\0','\0',...	(16 nulls)

MQNC_* (Name Count):

MQNC_MAX_NAMELIST_NAME_COUNT	256	X'00000100'

MQNPM_* (Nonpersistent Message Class):

MQNPM_CLASS_NORMAL	0	X'00000000'
MQNPM_CLASS_HIGH	10	X'0000000A'

MQNPMS_* (NonPersistent-Message Speeds):

MQNPMS_NORMAL	1	X'00000001'
MQNPMS_FAST	2	X'00000002'

MQNT_* (Namelist Types):

MQNT_NONE	0	X'00000000'
MQNT_Q	1	X'00000001'
MQNT_CLUSTER	2	X'00000002'
MQNT_AUTH_INFO	4	X'00000004'
MQNT_ALL	1001	X'000003E9'

MQNVS_* (Names for Name/Value String):

MQNVS_APPL_TYPE	"OPT_APP_GRPb"	
MQNVS_MSG_TYPE	"OPT_MSG_TYPEb"	

MQOA_* (Limits for Selectors for Object Attributes):

MQOA_FIRST	1	X'00000001'
MQOA_LAST	9000	X'00002328'

MQOD_* (Object descriptor structure):

MQOD_STRUC_ID	"ODbb"	
MQOD_STRUC_ID_ARRAY	'O','D','b','b'	
MQOD_VERSION_1	1	X'00000001'
MQOD_VERSION_2	2	X'00000002'
MQOD_VERSION_3	3	X'00000003'
MQOD_VERSION_4	4	X'00000004'
MQOD_CURRENT_VERSION	4	X'00000004'
MQOD_CURRENT_LENGTH	(value differs by platform or version)	

MQOII_* (Object Instance Identifier):

MQOII_NONE	X'00...00'	(24 nulls)
MQOII_NONE_ARRAY	'\0','\0',...	(24 nulls)

MQOL_* (Original Length):

MQOL_UNDEFINED	-1	X'FFFFFFFF'

MQOM_* (Obsolete Db2 Messages options on Inquire Group):

MQOM_NO	0	X'00000000'
MQOM_YES	1	X'00000001'

MQOO_* (Open Options):

MQOO_BIND_AS_Q_DEF	0	X'00000000'
MQOO_READ_AHEAD_AS_Q_DEF	0	X'00000000'
MQOO_INPUT_AS_Q_DEF	1	X'00000001'
MQOO_INPUT_SHARED	2	X'00000002'
MQOO_INPUT_EXCLUSIVE	4	X'00000004'
MQOO_BROWSE	8	X'00000008'
MQOO_OUTPUT	16	X'00000010'
MQOO_INQUIRE	32	X'00000020'
MQOO_SET	64	X'00000040'

MQOO_SAVE_ALL_CONTEXT	128	X'00000080'
MQOO_PASS_IDENTITY_CONTEXT	256	X'00000100'
MQOO_PASS_ALL_CONTEXT	512	X'00000200'
MQOO_SET_IDENTITY_CONTEXT	1024	X'00000400'
MQOO_SET_ALL_CONTEXT	2048	X'00000800'
MQOO_ALTERNATE_USER_AUTHORITY	4096	X'00001000'
MQOO_FAIL_IF_QUIESCING	8192	X'00002000'
MQOO_BIND_ON_OPEN	16384	X'00004000'
MQOO_BIND_NOT_FIXED	32768	X'00008000'
MQOO_CO_OP	131072	X'00020000'
MQOO_RESOLVE_LOCAL_TOPIC	262144	X'00040000'
MQOO_NO_READ_AHEAD	524288	X'00080000'
MQOO_READ_AHEAD	1048576	X'00100000'
MQOO_BIND_ON_GROUP	4194304	X'00400000'

MQOO_* (Following used in C++ only):

MQOO_RESOLVE_NAMES	65536	X'00010000'
MQOO_RESOLVE_LOCAL_Q	262144	X'00040000'

MQOP_* (Operation codes for MQCTL and MQCB):
Operation codes for MQCTL

MQOP_START	1	X'00000001'
MQOP_START_WAIT	2	X'00000002'
MQOP_STOP	4	X'00000004'

Operation codes for MQCB

MQOP_REGISTER	256	X'00000100'
MQOP_DEREGISTER	512	X'00000200'

Operation codes for MQCTL and MQCB

MQOP_SUSPEND	65536	X'00010000'
MQOP_RESUME	131072	X'00020000'

MQOPEN_* (Values related to MQOPEN_PRIV structure):

MQOPEN_PRIV_VERSION_1	1	X'00000001'
MQOPEN_PRIV_CURRENT_VERSION	1	X'00000001'

MQOPER_* (Activity Operations):

MQOPER_SYSTEM_FIRST	0	X'00000000'
MQOPER_UNKNOWN	0	X'00000000'
MQOPER_BROWSE	1	X'00000001'
MQOPER_DISCARD	2	X'00000002'
MQOPER_GET	3	X'00000003'
MQOPER_PUT	4	X'00000004'
MQOPER_PUT_REPLY	5	X'00000005'
MQOPER_PUT_REPORT	6	X'00000006'
MQOPER_RECEIVE	7	X'00000007'
MQOPER_SEND	8	X'00000008'
MQOPER_TRANSFORM	9	X'00000009'
MQOPER_PUBLISH	10	X'0000000A'
MQOPER_EXCLUDED_PUBLISH	11	X'0000000B'
MQOPER_DISCARDED_PUBLISH	12	X'0000000C'
MQOPER_SYSTEM_LAST	65535	X'0000FFFF'
MQOPER_APPL_FIRST	65536	X'00010000'
MQOPER_APPL_LAST	999999999	X'3B9AC9FF'

MQOT_* (Object Types and Extended Object Types):
Object Types

MQOT_NONE	0	X'00000000'
MQOT_Q	1	X'00000001'
MQOT_NAMELIST	2	X'00000002'
MQOT_PROCESS	3	X'00000003'
MQOT_STORAGE_CLASS	4	X'00000004'
MQOT_Q_MGR	5	X'00000005'
MQOT_CHANNEL	6	X'00000006'
MQOT_AUTH_INFO	7	X'00000007'
MQOT_TOPIC	8	X'00000008'
MQOT_CF_STRUC	10	X'0000000A'
MQOT_LISTENER	11	X'0000000B'
MQOT_SERVICE	12	X'0000000C'
MQOT_RESERVED_1	999	X'000003E7'

Extended Object Types

MQOT_ALL	1001	X'000003E9'
MQOT_ALIAS_Q	1002	X'000003EA'
MQOT_MODEL_Q	1003	X'000003EB'
MQOT_LOCAL_Q	1004	X'000003EC'
MQOT_REMOTE_Q	1005	X'000003ED'
MQOT_SENDER_CHANNEL	1007	X'000003EF'
MQOT_SERVER_CHANNEL	1008	X'000003F0'
MQOT_REQUESTER_CHANNEL	1009	X'000003F1'
MQOT_RECEIVER_CHANNEL	1010	X'000003F2'
MQOT_CURRENT_CHANNEL	1011	X'000003F3'
MQOT_SAVED_CHANNEL	1012	X'000003F4'
MQOT_SVRCONN_CHANNEL	1013	X'000003F5'
MQOT_CLNTCONN_CHANNEL	1014	X'000003F6'
MQOT_SHORT_CHANNEL	1015	X'000003F7'

MQPA_* (Put Authority):

MQPA_DEFAULT	1	X'00000001'
MQPA_CONTEXT	2	X'00000002'
MQPA_ONLY_MCA	3	X'00000003'
MQPA_ALTERNATE_OR_MCA	4	X'00000004'

MQPD_* (Property descriptor, support and context):
Property descriptor structure

MQPD_STRUC_ID	"PDbb"	
MQPD_STRUC_ID_ARRAY	'P','D','b','b'	
MQPD_VERSION_1	1	X'00000001'
MQPD_CURRENT_VERSION	1	X'00000001'

Property Descriptor Options

MQPD_NONE	0	X'00000000'

Property Support Options

MQPD_SUPPORT_OPTIONAL	1	X'00000001'
MQPD_SUPPORT_REQUIRED	1048576	X'00100000'
MQPD_SUPPORT_REQUIRED_IF_LOCAL	1024	X'00000400'
MQPD_REJECT_UNSUP_MASK	-1048576	X'FFF00000'
MQPD_ACCEPT_UNSUP_IF_XMIT_MASK	1047552	X'000FFC00'
MQPD_ACCEPT_UNSUP_MASK	1023	X'000003FF'

Property Context

MQPD_NO_CONTEXT	0	X'00000000'
MQPD_USER_CONTEXT	1	X'00000001'

MQPER_* (Persistence Values):

MQPER_PERSISTENCE_AS_PARENT	-1	X'FFFFFFFF'
MQPER_NOT_PERSISTENT	0	X'00000000'
MQPER_PERSISTENT	1	X'00000001'
MQPER_PERSISTENCE_AS_Q_DEF	2	X'00000002'
MQPER_PERSISTENCE_AS_TOPIC_DEF	2	X'00000002'

MQPL_* (Platforms):

MQPL_MVS	1	X'00000001'
MQPL_OS390	1	X'00000001'
MQPL_ZOS	1	X'00000001'
MQPL_OS2	2	X'00000002'
MQPL_AIX	3	X'00000003'
MQPL_UNIX	3	X'00000003'
MQPL_OS400	4	X'00000004'
MQPL_WINDOWS	5	X'00000005'
MQPL_WINDOWS_NT	11	X'0000000B'
MQPL_VMS	12	X'0000000C'
MQPL_NSK	13	X'0000000D'
MQPL_NSS	13	X'0000000D'
MQPL_OPEN_TP1	15	X'0000000F'
MQPL_VM	18	X'00000012'
MQPL_TPF	23	X'00000017'
MQPL_VSE	27	X'0000001B'
MQPL_NATIVE	1	X'00000001'

MQPMO_* (Put message options and structure for publish mask):
Put message options structure

MQPMO_STRUC_ID	"PMOb"	
MQPMO_STRUC_ID_ARRAY	'P','M','O','b'	
MQPMO_VERSION_1	1	X'00000001'
MQPMO_VERSION_2	2	X'00000002'
MQPMO_VERSION_3	3	X'00000003'
MQPMO_CURRENT_VERSION	3	X'00000003'
MQPMO_CURRENT_LENGTH	(value differs by platform or version)	

Put Message Options

MQPMO_SYNCPOINT	2	X'00000002'
MQPMO_NO_SYNCPOINT	4	X'00000004'
MQPMO_DEFAULT_CONTEXT	32	X'00000020'
MQPMO_NEW_MSG_ID	64	X'00000040'
MQPMO_NEW_CORREL_ID	128	X'00000080'
MQPMO_PASS_IDENTITY_CONTEXT	256	X'00000100'
MQPMO_PASS_ALL_CONTEXT	512	X'00000200'
MQPMO_SET_IDENTITY_CONTEXT	1024	X'00000400'
MQPMO_SET_ALL_CONTEXT	2048	X'00000800'
MQPMO_ALTERNATE_USER_AUTHORITY	4096	X'00001000'
MQPMO_FAIL_IF_QUIESCING	8192	X'00002000'
MQPMO_NO_CONTEXT	16384	X'00004000'
MQPMO_LOGICAL_ORDER	32768	X'00008000'
MQPMO_ASYNC_RESPONSE	65536	X'00010000'
MQPMO_SYNC_RESPONSE	131072	X'00020000'
MQPMO_RESOLVE_LOCAL_Q	262144	X'00040000'
MQPMO_RETAIN	2097152	X'00200000'
MQPMO_MD_FOR_OUTPUT_ONLY	8388608	X'00800000'
MQPMO_SCOPE_QMGR	67108864	X'04000000'
MQPMO_SUPPRESS_REPLYTO	134217728	X'08000000'
MQPMO_NOT_OWN_SUBS	268435456	X'10000000'
MQPMO_RESPONSE_AS_Q_DEF	0	X'00000000'
MQPMO_RESPONSE_AS_TOPIC_DEF	0	X'00000000'
MQPMO_NONE	0	X'00000000'

Put Message Options for publish mask

MQPMO_PUB_OPTIONS_MASK	2097152	X'00200000'

MQPMRF_* (Put Message Record Fields):

MQPMRF_MSG_ID	1	X'00000001'
MQPMRF_CORREL_ID	2	X'00000002'
MQPMRF_GROUP_ID	4	X'00000004'
MQPMRF_FEEDBACK	8	X'00000008'
MQPMRF_ACCOUNTING_TOKEN	16	X'00000010'
MQPMRF_NONE	0	X'00000000'

MQPO_* (Command format Purge Options):

MQPO_YES	1	X'00000001'
MQPO_NO	0	X'00000000'

MQPRI_* (Priority):

MQPRI_PRIORITY_AS_Q_DEF	-1	X'FFFFFFFF'
MQPRI_PRIORITY_AS_PARENT	-2	X'FFFFFFFE'
MQPRI_PRIORITY_AS_PUBLISHED	-3	X'FFFFFFFD'
MQPRI_PRIORITY_AS_TOPIC_DEF	-1	X'FFFFFFFF'

MQPROP_* (Queue and Channel Property Control Values and Maximum Properties Length):
Queue and Channel Property Control Values

MQPROP_COMPATIBILITY	0	X'00000000'
MQPROP_NONE	1	X'00000001'
MQPROP_ALL	2	X'00000002'
MQPROP_FORCE_MQRFH2	3	X'00000003'

Maximum Properties Length

MQPROP_UNRESTRICTED_LENGTH	-1	X'FFFFFFFF'

MQPRT_* (Put Response Values):

MQPRT_RESPONSE_AS_PARENT	0	X'00000000'
MQPRT_SYNC_RESPONSE	1	X'00000001'
MQPRT_ASYNC_RESPONSE	2	X'00000002'

MQPS_* (Publish/Subscribe):
Command format Publish/Subscribe Status

MQPS_STATUS_INACTIVE	0	X'00000000'
MQPS_STATUS_STARTING	1	X'00000001'
MQPS_STATUS_STOPPING	2	X'00000002'
MQPS_STATUS_ACTIVE	3	X'00000003'
MQPS_STATUS_COMPAT	4	X'00000004'
MQPS_STATUS_ERROR	5	X'00000005'
MQPS_STATUS_REFUSED	6	X'00000006'

Publish/Subscribe Tags as strings

MQPS_COMMAND	"MQPSCommand"	
MQPS_COMP_CODE	"MQPSCompCode"	
MQPS_CORREL_ID	"MQPSCorrelId"	
MQPS_DELETE_OPTIONS	"MQPSDelOpts"	
MQPS_ERROR_ID	"MQPSErrorId"	
MQPS_ERROR_POS	"MQPSErrorPos"	
MQPS_INTEGER_DATA	"MQPSIntData"	
MQPS_PARAMETER_ID	"MQPSParmId"	
MQPS_PUBLICATION_OPTIONS	"MQPSPubOpts"	
MQPS_PUBLISH_TIMESTAMP	"MQPSPubTime"	
MQPS_Q_MGR_NAME	"MQPSQMgrName"	
MQPS_Q_NAME	"MQPSQName"	
MQPS_REASON	"MQPSReason"	
MQPS_REASON_TEXT	"MQPSReasonText"	
MQPS_REGISTRATION_OPTIONS	"MQPSRegOpts"	
MQPS_SEQUENCE_NUMBER	"MQPSSeqNum"	
MQPS_STREAM_NAME	"MQPSStreamName"	
MQPS_STRING_DATA	"MQPSStringData"	
MQPS_SUBSCRIPTION_IDENTITY	"MQPSSubIdentity"	
MQPS_SUBSCRIPTION_NAME	"MQPSSubName"	
MQPS_SUBSCRIPTION_USER_DATA	"MQPSSubUserData"	
MQPS_TOPIC	"MQPSTopic"	
MQPS_USER_ID	"MQPSUserId"	

Publish/Subscribe Tags as blank-enclosed strings

MQPS_COMMAND_B	"bMQPSCommandb"	
MQPS_COMP_CODE_B	"bMQPSCompCodeb"	
MQPS_CORREL_ID_B	"bMQPSCorrelIdb"	
MQPS_DELETE_OPTIONS_B	"bMQPSDelOptsb"	
MQPS_ERROR_ID_B	"bMQPSErrorIdb"	
MQPS_ERROR_POS_B	"bMQPSErrorPosb"	
MQPS_INTEGER_DATA_B	"bMQPSIntDatab"	
MQPS_PARAMETER_ID_B	"bMQPSParmIdb"	
MQPS_PUBLICATION_OPTIONS_B	"bMQPSPubOptsb"	
MQPS_PUBLISH_TIMESTAMP_B	"bMQPSPubTimeb"	
MQPS_Q_MGR_NAME_B	"bMQPSQMgrNameb"	
MQPS_Q_NAME_B	"bMQPSQNameb"	
MQPS_REASON_B	"bMQPSReasonb"	
MQPS_REASON_TEXT_B	"bMQPSReasonTextb"	

MQPS_REGISTRATION_OPTIONS_B	"bMQPSRegOptsb"	
MQPS_SEQUENCE_NUMBER_B	"bMQPSSeqNumb"	
MQPS_STREAM_NAME_B	"bMQPSStreamNameb"	
MQPS_STRING_DATA_B	"bMQPSStringDatab"	
MQPS_SUBSCRIPTION_IDENTITY_B	"bMQPSSubIdentityb"	
MQPS_SUBSCRIPTION_NAME_B	"bMQPSSubNameb"	
MQPS_SUBSCRIPTION_USER_DATA_B	"bMQPSSubUserDatab"	
MQPS_TOPIC_B	"bMQPSTopicb"	
MQPS_USER_ID_B	"bMQPSUserIdb"	

Publish/Subscribe Command Tag Values as strings

MQPS_DELETE_PUBLICATION	"DeletePub"	
MQPS_DEREGISTER_PUBLISHER	"DeregPub"	
MQPS_DEREGISTER_SUBSCRIBER	"DeregSub"	
MQPS_PUBLISH	"Publish"	
MQPS_REGISTER_PUBLISHER	"RegPub"	
MQPS_REGISTER_SUBSCRIBER	"RegSub"	
MQPS_REQUEST_UPDATE	"ReqUpdate"	

Publish/Subscribe Command Tag Values as blank-enclosed strings

MQPS_DELETE_PUBLICATION_B	"bDeletePubb"	
MQPS_DEREGISTER_PUBLISHER_B	"bDeregPubb"	
MQPS_DEREGISTER_SUBSCRIBER_B	"bDeregSubb"	
MQPS_PUBLISH_B	"bPublishb"	
MQPS_REGISTER_PUBLISHER_B	"bRegPubb"	
MQPS_REGISTER_SUBSCRIBER_B	"bRegSubb"	
MQPS_REQUEST_UPDATE_B	"bReqUpdateb"	

Publish/Subscribe Options Tag Values as strings

MQPS_ADD_NAME	"AddName"	
MQPS_ANONYMOUS	"Anon"	
MQPS_CORREL_ID_AS_IDENTITY	"CorrelAsId"	
MQPS_DEREGISTER_ALL	"DeregAll"	
MQPS_DIRECT_REQUESTS	"DirectReq"	
MQPS_DUPLICATES_OK	"DupsOK"	
MQPS_FULL_RESPONSE	"FullResp"	
MQPS_INCLUDE_STREAM_NAME	"InclStreamName"	
MQPS_INFORM_IF_RETAINED	"InformIfRet"	

MQPS_IS_RETAINED_PUBLICATION	"IsRetainedPub"	
MQPS_JOIN_EXCLUSIVE	"JoinExcl"	
MQPS_JOIN_SHARED	"JoinShared"	
MQPS_LEAVE_ONLY	"LeaveOnly"	
MQPS_LOCAL	"Local"	
MQPS_LOCKED	"Locked"	
MQPS_NEW_PUBLICATIONS_ONLY	"NewPubsOnly"	
MQPS_NO_ALTERATION	"NoAlter"	
MQPS_NO_REGISTRATION	"NoReg"	
MQPS_NON_PERSISTENT	"NonPers"	
MQPS_NONE	"None"	
MQPS_OTHER_SUBSCRIBERS_ONLY	"OtherSubsOnly"	
MQPS_PERSISTENT	"Pers"	
MQPS_PERSISTENT_AS_PUBLISH	"PersAsPub"	
MQPS_PERSISTENT_AS_Q	"PersAsQueue"	
MQPS_PUBLISH_ON_REQUEST_ONLY	"PubOnReqOnly"	
MQPS_RETAIN_PUBLICATION	"RetainPub"	
MQPS_VARIABLE_USER_ID	"VariableUserId"	

Publish/Subscribe Options Tag Values as blank-enclosed strings

MQPS_ADD_NAME_B	"bAddNameb"	
MQPS_ANONYMOUS_B	"bAnonb"	
MQPS_CORREL_ID_AS_IDENTITY_B	"bCorrelAsIdb"	
MQPS_DEREGISTER_ALL_B	"bDeregAllb"	
MQPS_DIRECT_REQUESTS_B	"bDirectReqb"	
MQPS_DUPLICATES_OK_B	"bDupsOKb"	
MQPS_FULL_RESPONSE_B	"bFullRespb"	
MQPS_INCLUDE_STREAM_NAME_B	"bInclStreamNameb"	
MQPS_INFORM_IF_RETAINED_B	"bInformIfRetb"	
MQPS_IS_RETAINED_PUBLICATION_B	"bIsRetainedPubb"	
MQPS_JOIN_EXCLUSIVE_B	"bJoinExclb"	
MQPS_JOIN_SHARED_B	"bJoinSharedb"	
MQPS_LEAVE_ONLY_B	"bLeaveOnlyb"	
MQPS_LOCAL_B	"bLocalb"	
MQPS_LOCKED_B	"bLockedb"	
MQPS_NEW_PUBLICATIONS_ONLY_B	"bNewPubsOnlyb"	
MQPS_NO_ALTERATION_B	"bNoAlterb"	
MQPS_NO_REGISTRATION_B	"bNoRegb"	
MQPS_NON_PERSISTENT_B	"bNonPersb"	

MQPS_NONE_B	"bNoneb"	
MQPS_OTHER_SUBSCRIBERS_ONLY_B	"bOtherSubsOnlyb"	
MQPS_PERSISTENT_B	"bPersb"	
MQPS_PERSISTENT_AS_PUBLISH_B	"bPersAsPubb"	
MQPS_PERSISTENT_AS_Q_B	"bPersAsQueueb"	
MQPS_PUBLISH_ON_REQUEST_ONLY_B	"bPubOnReqOnlyb"	
MQPS_RETAIN_PUBLICATION_B	"bRetainPubb"	
MQPS_VARIABLE_USER_ID_B	"bVariableUserIdb"	

MQPSC_* (Publish/Subscribe Options Tag Publish/Subscribe Command Folder (psc) Tags):

MQPSC_FOLDER_VERSION	1	X'00000001'

MQPSC_* (Publish/Subscribe Options Tag Tag names):

MQPSC_COMMAND	"Command"	
MQPSC_REGISTRATION_OPTION	"RegOpt"	
MQPSC_PUBLICATION_OPTION	"PubOpt"	
MQPSC_DELETE_OPTION	"DelOpt"	
MQPSC_TOPIC	"Topic"	
MQPSC_SUBSCRIPTION_POINT	"SubPoint"	
MQPSC_FILTER	"Filter"	
MQPSC_Q_MGR_NAME	"QMgrName"	
MQPSC_Q_NAME	"QName"	
MQPSC_PUBLISH_TIMESTAMP	"PubTime"	
MQPSC_SEQUENCE_NUMBER	"SeqNum"	
MQPSC_SUBSCRIPTION_NAME	"SubName"	
MQPSC_SUBSCRIPTION_IDENTITY	"SubIdentity"	
MQPSC_SUBSCRIPTION_USER_DATA	"SubUserData"	
MQPSC_CORREL_ID	"CorrelId"	

MQPSC_* (Publish/Subscribe Options Tag XML tag names):

MQPSC_COMMAND_B	"<Command>"	
MQPSC_COMMAND_E	"</Command>"	
MQPSC_REGISTRATION_OPTION_B	"<RegOpt>"	
MQPSC_REGISTRATION_OPTION_E	"</RegOpt>"	
MQPSC_PUBLICATION_OPTION_B	"<PubOpt>"	
MQPSC_PUBLICATION_OPTION_E	"</PubOpt>"	
MQPSC_DELETE_OPTION_B	"<DelOpt>"	
MQPSC_DELETE_OPTION_E	"</DelOpt>"	

MQPSC_TOPIC_B	"<Topic>"	
MQPSC_TOPIC_E	"</Topic>"	
MQPSC_SUBSCRIPTION_POINT_B	"<SubPoint>"	
MQPSC_SUBSCRIPTION_POINT_E	"</SubPoint>"	
MQPSC_FILTER_B	"<Filter>"	
MQPSC_FILTER_E	"</Filter>"	
MQPSC_Q_MGR_NAME_B	"<QMgrName>"	
MQPSC_Q_MGR_NAME_E	"</QMgrName>"	
MQPSC_Q_NAME_B	"<QName>"	
MQPSC_Q_NAME_E	"</QName>"	
MQPSC_PUBLISH_TIMESTAMP_B	"<PubTime>"	
MQPSC_PUBLISH_TIMESTAMP_E	"</PubTime>"	
MQPSC_SEQUENCE_NUMBER_B	"<SeqNum>"	
MQPSC_SEQUENCE_NUMBER_E	"</SeqNum>"	
MQPSC_SUBSCRIPTION_NAME_B	"<SubName>"	
MQPSC_SUBSCRIPTION_NAME_E	"</SubName>"	
MQPSC_SUBSCRIPTION_IDENTITY_B	"<SubIdentity>"	
MQPSC_SUBSCRIPTION_IDENTITY_E	"</SubIdentity>"	
MQPSC_SUBSCRIPTION_USER_DATA_B	"<SubUserData>"	
MQPSC_SUBSCRIPTION_USER_DATA_E	"</SubUserData>"	
MQPSC_CORREL_ID_B	"<CorrelId>"	
MQPSC_CORREL_ID_E	"</CorrelId>"	

MQPSC_* (Publish/Subscribe Options Tag Values as strings):

MQPSC_DELETE_PUBLICATION	"DeletePub"	
MQPSC_DEREGISTER_SUBSCRIBER	"DeregSub"	
MQPSC_PUBLISH	"Publish"	
MQPSC_REGISTER_SUBSCRIBER	"RegSub"	
MQPSC_REQUEST_UPDATE	"ReqUpdate"	

MQPSC_* (Publish/Subscribe Options Tag Values as strings):

MQPSC_ADD_NAME	"AddName"	
MQPSC_CORREL_ID_AS_IDENTITY	"CorrelAsId"	
MQPSC_DEREGISTER_ALL	"DeregAll"	
MQPSC_DUPLICATES_OK	"DupsOK"	
MQPSC_FULL_RESPONSE	"FullResp"	
MQPSC_INFORM_IF_RETAINED	"InformIfRet"	
MQPSC_IS_RETAINED_PUB	"IsRetainedPub"	

MQPSC_JOIN_SHARED	"JoinShared"	
MQPSC_JOIN_EXCLUSIVE	"JoinExcl"	
MQPSC_LEAVE_ONLY	"LeaveOnly"	
MQPSC_LOCAL	"Local"	
MQPSC_LOCKED	"Locked"	
MQPSC_NEW_PUBS_ONLY	"NewPubsOnly"	
MQPSC_NO_ALTERATION	"NoAlter"	
MQPSC_NON_PERSISTENT	"NonPers"	
MQPSC_OTHER_SUBS_ONLY	"OtherSubsOnly"	
MQPSC_PERSISTENT	"Pers"	
MQPSC_PERSISTENT_AS_PUBLISH	"PersAsPub"	
MQPSC_PERSISTENT_AS_Q	"PersAsQueue"	
MQPSC_NONE	"None"	
MQPSC_PUB_ON_REQUEST_ONLY	"PubOnReqOnly"	
MQPSC_RETAIN_PUB	"RetainPub"	
MQPSC_VARIABLE_USER_ID	"VariableUserId"	

MQPSCR_* (Publish/Subscribe Options):

Publish/Subscribe Options Tag Publish/Subscribe Response Folder (pscr) Tags

MQPSCR_FOLDER_VERSION	1	X'00000001'

Publish/Subscribe Options Tag Tag names

MQPSCR_COMPLETION	"Completion"	
MQPSCR_RESPONSE	"Response"	
MQPSCR_REASON	"Reason"	

Publish/Subscribe Options Tag XML tag names

MQPSCR_COMPLETION_B	"<Completion>"	
MQPSCR_COMPLETION_E	"</Completion>"	
MQPSCR_RESPONSE_B	"<Response>"	
MQPSCR_RESPONSE_E	"</Response>"	
MQPSCR_REASON_B	"<Reason>"	
MQPSCR_REASON_E	"</Reason>"	

Publish/Subscribe Options Tag Tag values

MQPSCR_OK	"ok"	
MQPSCR_WARNING	"warning"	
MQPSCR_ERROR	"error"	

MQPSM_* (Pub/Sub Mode):

MQPSM_DISABLED	0	X'00000000'
MQPSM_COMPAT	1	X'00000001'
MQPSM_ENABLED	2	X'00000002'

MQPSPROP_* (Pub/Sub Message Properties):

MQPSPROP_NONE	0	X'00000000'
MQPSPROP_COMPAT	1	X'00000001'
MQPSPROP_RFH2	2	X'00000002'
MQPSPROP_MSGPROP	3	X'00000003'

MQPSST_* (Command format Pub/Sub Status Type):

MQPSST_ALL	0	X'00000000'
MQPSST_LOCAL	1	X'00000001'
MQPSST_PARENT	2	X'00000002'
MQPSST_CHILD	3	X'00000003'

MQPUBO_* (Publish/Subscribe Publication Options):

MQPUBO_NONE	0	X'00000000'
MQPUBO_CORREL_ID_AS_IDENTITY	1	X'00000001'
MQPUBO_RETAIN_PUBLICATION	2	X'00000002'
MQPUBO_OTHER_SUBSCRIBERS_ONLY	4	X'00000004'
MQPUBO_NO_REGISTRATION	8	X'00000008'
MQPUBO_IS_RETAINED_PUBLICATION	16	X'00000010'

MQPXP_* (Publish/subscribe routing exit parameter structure):

MQPXP_STRUC_ID	"PXPb"	
MQPXP_STRUC_ID_ARRAY	'P','X','P','b'	
MQPXP_VERSION_1	1	X'00000001'
MQPXP_CURRENT_VERSION	1	X'00000001'

MQQA_* (Queue attributes):
Inhibit Get Values

MQQA_GET_INHIBITED	1	X'00000001'
MQQA_GET_ALLOWED	0	X'00000000'

Inhibit Put Values

MQQA_PUT_INHIBITED	1	X'00000001'
MQQA_PUT_ALLOWED	0	X'00000000'

Queue Shareability

MQQA_SHAREABLE	1	X'00000001'
MQQA_NOT_SHAREABLE	0	X'00000000'

Back-Out Hardening

MQQA_BACKOUT_HARDENED	1	X'00000001'
MQQA_BACKOUT_NOT_HARDENED	0	X'00000000'

MQQDT_* (Queue Definition Types):

MQQDT_PREDEFINED	1	X'00000001'
MQQDT_PERMANENT_DYNAMIC	2	X'00000002'
MQQDT_TEMPORARY_DYNAMIC	3	X'00000003'
MQQDT_SHARED_DYNAMIC	4	X'00000004'

MQQF_* (Queue Flags):

MQQF_LOCAL_Q	1	X'00000001'
MQQF_CLWL_USEQ_ANY	64	X'00000040'
MQQF_CLWL_USEQ_LOCAL	128	X'00000080'

MQQMDT_* (Command format Queue Manager Definition Types):

MQQMDT_EXPLICIT_CLUSTER_SENDER	1	X'00000001'
MQQMDT_AUTO_CLUSTER_SENDER	2	X'00000002'
MQQMDT_AUTO_EXP_CLUSTER_SENDER	4	X'00000004'
MQQMDT_CLUSTER_RECEIVER	3	X'00000003'

MQQMF_* (Queue Manager Flags):

MQQMF_REPOSITORY_Q_MGR	2	X'00000002'
MQQMF_CLUSSDR_USER_DEFINED	8	X'00000008'
MQQMF_CLUSSDR_AUTO_DEFINED	16	X'00000010'
MQQMF_AVAILABLE	32	X'00000020'

MQQMFAC_* (Command format Queue Manager Facility):

MQQMFAC_IMS_BRIDGE	1	X'00000001'
MQQMFAC_DB2	2	X'00000002'

MQQMSTA_* (Command format Queue Manager Status):

MQQMSTA_STARTING	1	X'00000001'
MQQMSTA_RUNNING	2	X'00000002'
MQQMSTA_QUIESCING	3	X'00000003'

MQQMT_* (Command format Queue Manager Types):

MQQMT_NORMAL	0	X'00000000'
MQQMT_REPOSITORY	1	X'00000001'

MQQO_* (Command format Quiesce Options):

MQQO_YES	1	X'00000001'
MQQO_NO	0	X'00000000'

MQQSGD_* (Queue Sharing Group Dispositions):

MQQSGD_ALL	-1	X'FFFFFFFF'
MQQSGD_Q_MGR	0	X'00000000'
MQQSGD_COPY	1	X'00000001'
MQQSGD_SHARED	2	X'00000002'
MQQSGD_GROUP	3	X'00000003'
MQQSGD_PRIVATE	4	X'00000004'
MQQSGD_LIVE	6	X'00000006'

MQQSGS_* (Command format QSG Status):

MQQSGS_UNKNOWN	0	X'00000000'
MQQSGS_CREATED	1	X'00000001'
MQQSGS_ACTIVE	2	X'00000002'
MQQSGS_INACTIVE	3	X'00000003'
MQQSGS_FAILED	4	X'00000004'
MQQSGS_PENDING	5	X'00000005'

MQQSIE_* (Command format Queue Service-Interval Events):

MQQSIE_NONE	0	X'00000000'
MQQSIE_HIGH	1	X'00000001'
MQQSIE_OK	2	X'00000002'

MQQSO_* (Command format Queue Status Open Options for SET, BROWSE, INPUT):

MQQSO_NO	0	X'00000000'
MQQSO_YES	1	X'00000001'
MQQSO_SHARED	1	X'00000001'
MQQSO_EXCLUSIVE	2	X'00000002'

MQQSOT_* (Command format Queue Status Open Types):

MQQSOT_ALL	1	X'00000001'
MQQSOT_INPUT	2	X'00000002'
MQQSOT_OUTPUT	3	X'00000003'

MQQSUM_* (Command format Queue Status Uncommitted Messages):

MQQSUM_YES	1	X'00000001'
MQQSUM_NO	0	X'00000000'

MQQT_* (Queue Types and Extended Queue Types):
Queue Types

MQQT_LOCAL	1	X'00000001'
MQQT_MODEL	2	X'00000002'
MQQT_ALIAS	3	X'00000003'
MQQT_REMOTE	6	X'00000006'
MQQT_CLUSTER	7	X'00000007'

Extended Queue Types

MQQT_ALL	1001	X'000003E9'

MQRC_* (Reason Codes):

MQRC_NONE	0	X'00000000'
MQRC_APPL_FIRST	900	X'00000384'
MQRC_APPL_LAST	999	X'000003E7'
MQRC_ALIAS_BASE_Q_TYPE_ERROR	2001	X'000007D1'
MQRC_ALREADY_CONNECTED	2002	X'000007D2'
MQRC_BACKED_OUT	2003	X'000007D3'
MQRC_BUFFER_ERROR	2004	X'000007D4'

MQRC_BUFFER_LENGTH_ERROR	2005	X'000007D5'
MQRC_CHAR_ATTR_LENGTH_ERROR	2006	X'000007D6'
MQRC_CHAR_ATTRS_ERROR	2007	X'000007D7'
MQRC_CHAR_ATTRS_TOO_SHORT	2008	X'000007D8'
MQRC_CONNECTION_BROKEN	2009	X'000007D9'
MQRC_DATA_LENGTH_ERROR	2010	X'000007DA'
MQRC_DYNAMIC_Q_NAME_ERROR	2011	X'000007DB'
MQRC_ENVIRONMENT_ERROR	2012	X'000007DC'
MQRC_EXPIRY_ERROR	2013	X'000007DD'
MQRC_FEEDBACK_ERROR	2014	X'000007DE'
MQRC_GET_INHIBITED	2016	X'000007E0'
MQRC_HANDLE_NOT_AVAILABLE	2017	X'000007E1'
MQRC_HCONN_ERROR	2018	X'000007E2'
MQRC_HOBJ_ERROR	2019	X'000007E3'
MQRC_INHIBIT_VALUE_ERROR	2020	X'000007E4'
MQRC_INT_ATTR_COUNT_ERROR	2021	X'000007E5'
MQRC_INT_ATTR_COUNT_TOO_SMALL	2022	X'000007E6'
MQRC_INT_ATTRS_ARRAY_ERROR	2023	X'000007E7'
MQRC_SYNCPOINT_LIMIT_REACHED	2024	X'000007E8'
MQRC_MAX_CONNS_LIMIT_REACHED	2025	X'000007E9'
MQRC_MD_ERROR	2026	X'000007EA'
MQRC_MISSING_REPLY_TO_Q	2027	X'000007EB'
MQRC_MSG_TYPE_ERROR	2029	X'000007ED'
MQRC_MSG_TOO_BIG_FOR_Q	2030	X'000007EE'
MQRC_MSG_TOO_BIG_FOR_Q_MGR	2031	X'000007EF'
MQRC_NO_MSG_AVAILABLE	2033	X'000007F1'
MQRC_NO_MSG_UNDER_CURSOR	2034	X'000007F2'
MQRC_NOT_AUTHORIZED	2035	X'000007F3'
MQRC_NOT_OPEN_FOR_BROWSE	2036	X'000007F4'
MQRC_NOT_OPEN_FOR_INPUT	2037	X'000007F5'
MQRC_NOT_OPEN_FOR_INQUIRE	2038	X'000007F6'
MQRC_NOT_OPEN_FOR_OUTPUT	2039	X'000007F7'
MQRC_NOT_OPEN_FOR_SET	2040	X'000007F8'
MQRC_OBJECT_CHANGED	2041	X'000007F9'
MQRC_OBJECT_IN_USE	2042	X'000007FA'
MQRC_OBJECT_TYPE_ERROR	2043	X'000007FB'
MQRC_OD_ERROR	2044	X'000007FC'
MQRC_OPTION_NOT_VALID_FOR_TYPE	2045	X'000007FD'
MQRC_OPTIONS_ERROR	2046	X'000007FE'
MQRC_PERSISTENCE_ERROR	2047	X'000007FF'
MQRC_PERSISTENT_NOT_ALLOWED	2048	X'00000800'
MQRC_PRIORITY_EXCEEDS_MAXIMUM	2049	X'00000801'

MQRC_PRIORITY_ERROR	2050	X'00000802'
MQRC_PUT_INHIBITED	2051	X'00000803'
MQRC_Q_DELETED	2052	X'00000804'
MQRC_Q_FULL	2053	X'00000805'
MQRC_Q_NOT_EMPTY	2055	X'00000807'
MQRC_Q_SPACE_NOT_AVAILABLE	2056	X'00000808'
MQRC_Q_TYPE_ERROR	2057	X'00000809'
MQRC_Q_MGR_NAME_ERROR	2058	X'0000080A'
MQRC_Q_MGR_NOT_AVAILABLE	2059	X'0000080B'
MQRC_REPORT_OPTIONS_ERROR	2061	X'0000080D'
MQRC_SECOND_MARK_NOT_ALLOWED	2062	X'0000080E'
MQRC_SECURITY_ERROR	2063	X'0000080F'
MQRC_SELECTOR_COUNT_ERROR	2065	X'00000811'
MQRC_SELECTOR_LIMIT_EXCEEDED	2066	X'00000812'
MQRC_SELECTOR_ERROR	2067	X'00000813'
MQRC_SELECTOR_NOT_FOR_TYPE	2068	X'00000814'
MQRC_SIGNAL_OUTSTANDING	2069	X'00000815'
MQRC_SIGNAL_REQUEST_ACCEPTED	2070	X'00000816'
MQRC_STORAGE_NOT_AVAILABLE	2071	X'00000817'
MQRC_SYNCPOINT_NOT_AVAILABLE	2072	X'00000818'
MQRC_TRIGGER_CONTROL_ERROR	2075	X'0000081B'
MQRC_TRIGGER_DEPTH_ERROR	2076	X'0000081C'
MQRC_TRIGGER_MSG_PRIORITY_ERR	2077	X'0000081D'
MQRC_TRIGGER_TYPE_ERROR	2078	X'0000081E'
MQRC_TRUNCATED_MSG_ACCEPTED	2079	X'0000081F'
MQRC_TRUNCATED_MSG_FAILED	2080	X'00000820'
MQRC_UNKNOWN_ALIAS_BASE_Q	2082	X'00000822'
MQRC_UNKNOWN_OBJECT_NAME	2085	X'00000825'
MQRC_UNKNOWN_OBJECT_Q_MGR	2086	X'00000826'
MQRC_UNKNOWN_REMOTE_Q_MGR	2087	X'00000827'
MQRC_WAIT_INTERVAL_ERROR	2090	X'0000082A'
MQRC_XMIT_Q_TYPE_ERROR	2091	X'0000082B'
MQRC_XMIT_Q_USAGE_ERROR	2092	X'0000082C'
MQRC_NOT_OPEN_FOR_PASS_ALL	2093	X'0000082D'
MQRC_NOT_OPEN_FOR_PASS_IDENT	2094	X'0000082E'
MQRC_NOT_OPEN_FOR_SET_ALL	2095	X'0000082F'
MQRC_NOT_OPEN_FOR_SET_IDENT	2096	X'00000830'
MQRC_CONTEXT_HANDLE_ERROR	2097	X'00000831'
MQRC_CONTEXT_NOT_AVAILABLE	2098	X'00000832'
MQRC_SIGNAL1_ERROR	2099	X'00000833'
MQRC_OBJECT_ALREADY_EXISTS	2100	X'00000834'
MQRC_OBJECT_DAMAGED	2101	X'00000835'

MQRC_RESOURCE_PROBLEM	2102	X'00000836'
MQRC_ANOTHER_Q_MGR_CONNECTED	2103	X'00000837'
MQRC_UNKNOWN_REPORT_OPTION	2104	X'00000838'
MQRC_STORAGE_CLASS_ERROR	2105	X'00000839'
MQRC_COD_NOT_VALID_FOR_XCF_Q	2106	X'0000083A'
MQRC_XWAIT_CANCELED	2107	X'0000083B'
MQRC_XWAIT_ERROR	2108	X'0000083C'
MQRC_SUPPRESSED_BY_EXIT	2109	X'0000083D'
MQRC_FORMAT_ERROR	2110	X'0000083E'
MQRC_SOURCE_CCSID_ERROR	2111	X'0000083F'
MQRC_SOURCE_INTEGER_ENC_ERROR	2112	X'00000840'
MQRC_SOURCE_DECIMAL_ENC_ERROR	2113	X'00000841'
MQRC_SOURCE_FLOAT_ENC_ERROR	2114	X'00000842'
MQRC_TARGET_CCSID_ERROR	2115	X'00000843'
MQRC_TARGET_INTEGER_ENC_ERROR	2116	X'00000844'
MQRC_TARGET_DECIMAL_ENC_ERROR	2117	X'00000845'
MQRC_TARGET_FLOAT_ENC_ERROR	2118	X'00000846'
MQRC_NOT_CONVERTED	2119	X'00000847'
MQRC_CONVERTED_MSG_TOO_BIG	2120	X'00000848'
MQRC_TRUNCATED	2120	X'00000848'
MQRC_NO_EXTERNAL_PARTICIPANTS	2121	X'00000849'
MQRC_PARTICIPANT_NOT_AVAILABLE	2122	X'0000084A'
MQRC_OUTCOME_MIXED	2123	X'0000084B'
MQRC_OUTCOME_PENDING	2124	X'0000084C'
MQRC_BRIDGE_STARTED	2125	X'0000084D'
MQRC_BRIDGE_STOPPED	2126	X'0000084E'
MQRC_ADAPTER_STORAGE_SHORTAGE	2127	X'0000084F'
MQRC_UOW_IN_PROGRESS	2128	X'00000850'
MQRC_ADAPTER_CONN_LOAD_ERROR	2129	X'00000851'
MQRC_ADAPTER_SERV_LOAD_ERROR	2130	X'00000852'
MQRC_ADAPTER_DEFS_ERROR	2131	X'00000853'
MQRC_ADAPTER_DEFS_LOAD_ERROR	2132	X'00000854'
MQRC_ADAPTER_CONV_LOAD_ERROR	2133	X'00000855'
MQRC_BO_ERROR	2134	X'00000856'
MQRC_DH_ERROR	2135	X'00000857'
MQRC_MULTIPLE_REASONS	2136	X'00000858'
MQRC_OPEN_FAILED	2137	X'00000859'
MQRC_ADAPTER_DISC_LOAD_ERROR	2138	X'0000085A'
MQRC_CNO_ERROR	2139	X'0000085B'
MQRC_CICS_WAIT_FAILED	2140	X'0000085C'
MQRC_DLH_ERROR	2141	X'0000085D'
MQRC_HEADER_ERROR	2142	X'0000085E'

MQRC_SOURCE_LENGTH_ERROR	2143	X'0000085F'
MQRC_TARGET_LENGTH_ERROR	2144	X'00000860'
MQRC_SOURCE_BUFFER_ERROR	2145	X'00000861'
MQRC_TARGET_BUFFER_ERROR	2146	X'00000862'
MQRC_IIH_ERROR	2148	X'00000864'
MQRC_PCF_ERROR	2149	X'00000865'
MQRC_DBCS_ERROR	2150	X'00000866'
MQRC_OBJECT_NAME_ERROR	2152	X'00000868'
MQRC_OBJECT_Q_MGR_NAME_ERROR	2153	X'00000869'
MQRC_RECS_PRESENT_ERROR	2154	X'0000086A'
MQRC_OBJECT_RECORDS_ERROR	2155	X'0000086B'
MQRC_RESPONSE_RECORDS_ERROR	2156	X'0000086C'
MQRC_ASID_MISMATCH	2157	X'0000086D'
MQRC_PMO_RECORD_FLAGS_ERROR	2158	X'0000086E'
MQRC_PUT_MSG_RECORDS_ERROR	2159	X'0000086F'
MQRC_CONN_ID_IN_USE	2160	X'00000870'
MQRC_Q_MGR_QUIESCING	2161	X'00000871'
MQRC_Q_MGR_STOPPING	2162	X'00000872'
MQRC_DUPLICATE_RECOV_COORD	2163	X'00000873'
MQRC_PMO_ERROR	2173	X'0000087D'
MQRC_API_EXIT_NOT_FOUND	2182	X'00000886'
MQRC_API_EXIT_LOAD_ERROR	2183	X'00000887'
MQRC_REMOTE_Q_NAME_ERROR	2184	X'00000888'
MQRC_INCONSISTENT_PERSISTENCE	2185	X'00000889'
MQRC_GMO_ERROR	2186	X'0000088A'
MQRC_CICS_BRIDGE_RESTRICTION	2187	X'0000088B'
MQRC_STOPPED_BY_CLUSTER_EXIT	2188	X'0000088C'
MQRC_CLUSTER_RESOLUTION_ERROR	2189	X'0000088D'
MQRC_CONVERTED_STRING_TOO_BIG	2190	X'0000088E'
MQRC_TMC_ERROR	2191	X'0000088F'
MQRC_PAGESET_FULL	2192	X'00000890'
MQRC_STORAGE_MEDIUM_FULL	2192	X'00000890'
MQRC_PAGESET_ERROR	2193	X'00000891'
MQRC_NAME_NOT_VALID_FOR_TYPE	2194	X'00000892'
MQRC_UNEXPECTED_ERROR	2195	X'00000893'
MQRC_UNKNOWN_XMIT_Q	2196	X'00000894'
MQRC_UNKNOWN_DEF_XMIT_Q	2197	X'00000895'
MQRC_DEF_XMIT_Q_TYPE_ERROR	2198	X'00000896'
MQRC_DEF_XMIT_Q_USAGE_ERROR	2199	X'00000897'
MQRC_MSG_MARKED_BROWSE_CO_OP	2200	X'00000898'
MQRC_NAME_IN_USE	2201	X'00000899'
MQRC_CONNECTION_QUIESCING	2202	X'0000089A'

MQRC_CONNECTION_STOPPING	2203	X'0000089B'
MQRC_ADAPTER_NOT_AVAILABLE	2204	X'0000089C'
MQRC_MSG_ID_ERROR	2206	X'0000089E'
MQRC_CORREL_ID_ERROR	2207	X'0000089F'
MQRC_FILE_SYSTEM_ERROR	2208	X'000008A0'
MQRC_NO_MSG_LOCKED	2209	X'000008A1'
MQRC_SOAP_DOTNET_ERROR	2210	X'000008A2'
MQRC_SOAP_AXIS_ERROR	2211	X'000008A3'
MQRC_SOAP_URL_ERROR	2212	X'000008A4'
MQRC_FILE_NOT_AUDITED	2216	X'000008A8'
MQRC_CONNECTION_NOT_AUTHORIZED	2217	X'000008A9'
MQRC_MSG_TOO_BIG_FOR_CHANNEL	2218	X'000008AA'
MQRC_CALL_IN_PROGRESS	2219	X'000008AB'
MQRC_RMH_ERROR	2220	X'000008AC'
MQRC_Q_MGR_ACTIVE	2222	X'000008AE'
MQRC_Q_MGR_NOT_ACTIVE	2223	X'000008AF'
MQRC_Q_DEPTH_HIGH	2224	X'000008B0'
MQRC_Q_DEPTH_LOW	2225	X'000008B1'
MQRC_Q_SERVICE_INTERVAL_HIGH	2226	X'000008B2'
MQRC_Q_SERVICE_INTERVAL_OK	2227	X'000008B3'
MQRC_RFH_HEADER_FIELD_ERROR	2228	X'000008B4'
MQRC_RAS_PROPERTY_ERROR	2229	X'000008B5'
MQRC_UNIT_OF_WORK_NOT_STARTED	2232	X'000008B8'
MQRC_CHANNEL_AUTO_DEF_OK	2233	X'000008B9'
MQRC_CHANNEL_AUTO_DEF_ERROR	2234	X'000008BA'
MQRC_CFH_ERROR	2235	X'000008BB'
MQRC_CFIL_ERROR	2236	X'000008BC'
MQRC_CFIN_ERROR	2237	X'000008BD'
MQRC_CFSL_ERROR	2238	X'000008BE'
MQRC_CFST_ERROR	2239	X'000008BF'
MQRC_INCOMPLETE_GROUP	2241	X'000008C1'
MQRC_INCOMPLETE_MSG	2242	X'000008C2'
MQRC_INCONSISTENT_CCSIDS	2243	X'000008C3'
MQRC_INCONSISTENT_ENCODINGS	2244	X'000008C4'
MQRC_INCONSISTENT_UOW	2245	X'000008C5'
MQRC_INVALID_MSG_UNDER_CURSOR	2246	X'000008C6'
MQRC_MATCH_OPTIONS_ERROR	2247	X'000008C7'
MQRC_MDE_ERROR	2248	X'000008C8'
MQRC_MSG_FLAGS_ERROR	2249	X'000008C9'
MQRC_MSG_SEQ_NUMBER_ERROR	2250	X'000008CA'
MQRC_OFFSET_ERROR	2251	X'000008CB'
MQRC_ORIGINAL_LENGTH_ERROR	2252	X'000008CC'

MQRC_SEGMENT_LENGTH_ZERO	2253	X'000008CD'
MQRC_UOW_NOT_AVAILABLE	2255	X'000008CF'
MQRC_WRONG_GMO_VERSION	2256	X'000008D0'
MQRC_WRONG_MD_VERSION	2257	X'000008D1'
MQRC_GROUP_ID_ERROR	2258	X'000008D2'
MQRC_INCONSISTENT_BROWSE	2259	X'000008D3'
MQRC_XQH_ERROR	2260	X'000008D4'
MQRC_SRC_ENV_ERROR	2261	X'000008D5'
MQRC_SRC_NAME_ERROR	2262	X'000008D6'
MQRC_DEST_ENV_ERROR	2263	X'000008D7'
MQRC_DEST_NAME_ERROR	2264	X'000008D8'
MQRC_TM_ERROR	2265	X'000008D9'
MQRC_CLUSTER_EXIT_ERROR	2266	X'000008DA'
MQRC_CLUSTER_EXIT_LOAD_ERROR	2267	X'000008DB'
MQRC_CLUSTER_PUT_INHIBITED	2268	X'000008DC'
MQRC_CLUSTER_RESOURCE_ERROR	2269	X'000008DD'
MQRC_NO_DESTINATIONS_AVAILABLE	2270	X'000008DE'
MQRC_CONN_TAG_IN_USE	2271	X'000008DF'
MQRC_PARTIALLY_CONVERTED	2272	X'000008E0'
MQRC_CONNECTION_ERROR	2273	X'000008E1'
MQRC_OPTION_ENVIRONMENT_ERROR	2274	X'000008E2'
MQRC_CD_ERROR	2277	X'000008E5'
MQRC_CLIENT_CONN_ERROR	2278	X'000008E6'
MQRC_CHANNEL_STOPPED_BY_USER	2279	X'000008E7'
MQRC_HCONFIG_ERROR	2280	X'000008E8'
MQRC_FUNCTION_ERROR	2281	X'000008E9'
MQRC_CHANNEL_STARTED	2282	X'000008EA'
MQRC_CHANNEL_STOPPED	2283	X'000008EB'
MQRC_CHANNEL_CONV_ERROR	2284	X'000008EC'
MQRC_SERVICE_NOT_AVAILABLE	2285	X'000008ED'
MQRC_INITIALIZATION_FAILED	2286	X'000008EE'
MQRC_TERMINATION_FAILED	2287	X'000008EF'
MQRC_UNKNOWN_Q_NAME	2288	X'000008F0'
MQRC_SERVICE_ERROR	2289	X'000008F1'
MQRC_Q_ALREADY_EXISTS	2290	X'000008F2'
MQRC_USER_ID_NOT_AVAILABLE	2291	X'000008F3'
MQRC_UNKNOWN_ENTITY	2292	X'000008F4'
MQRC_UNKNOWN_AUTH_ENTITY	2293	X'000008F5'
MQRC_UNKNOWN_REF_OBJECT	2294	X'000008F6'
MQRC_CHANNEL_ACTIVATED	2295	X'000008F7'
MQRC_CHANNEL_NOT_ACTIVATED	2296	X'000008F8'
MQRC_UOW_CANCELED	2297	X'000008F9'

MQRC_FUNCTION_NOT_SUPPORTED	2298	X'000008FA'
MQRC_SELECTOR_TYPE_ERROR	2299	X'000008FB'
MQRC_COMMAND_TYPE_ERROR	2300	X'000008FC'
MQRC_MULTIPLE_INSTANCE_ERROR	2301	X'000008FD'
MQRC_SYSTEM_ITEM_NOT_ALTERABLE	2302	X'000008FE'
MQRC_BAG_CONVERSION_ERROR	2303	X'000008FF'
MQRC_SELECTOR_OUT_OF_RANGE	2304	X'00000900'
MQRC_SELECTOR_NOT_UNIQUE	2305	X'00000901'
MQRC_INDEX_NOT_PRESENT	2306	X'00000902'
MQRC_STRING_ERROR	2307	X'00000903'
MQRC_ENCODING_NOT_SUPPORTED	2308	X'00000904'
MQRC_SELECTOR_NOT_PRESENT	2309	X'00000905'
MQRC_OUT_SELECTOR_ERROR	2310	X'00000906'
MQRC_STRING_TRUNCATED	2311	X'00000907'
MQRC_SELECTOR_WRONG_TYPE	2312	X'00000908'
MQRC_INCONSISTENT_ITEM_TYPE	2313	X'00000909'
MQRC_INDEX_ERROR	2314	X'0000090A'
MQRC_SYSTEM_BAG_NOT_ALTERABLE	2315	X'0000090B'
MQRC_ITEM_COUNT_ERROR	2316	X'0000090C'
MQRC_FORMAT_NOT_SUPPORTED	2317	X'0000090D'
MQRC_SELECTOR_NOT_SUPPORTED	2318	X'0000090E'
MQRC_ITEM_VALUE_ERROR	2319	X'0000090F'
MQRC_HBAG_ERROR	2320	X'00000910'
MQRC_PARAMETER_MISSING	2321	X'00000911'
MQRC_CMD_SERVER_NOT_AVAILABLE	2322	X'00000912'
MQRC_STRING_LENGTH_ERROR	2323	X'00000913'
MQRC_INQUIRY_COMMAND_ERROR	2324	X'00000914'
MQRC_NESTED_BAG_NOT_SUPPORTED	2325	X'00000915'
MQRC_BAG_WRONG_TYPE	2326	X'00000916'
MQRC_ITEM_TYPE_ERROR	2327	X'00000917'
MQRC_SYSTEM_BAG_NOT_DELETABLE	2328	X'00000918'
MQRC_SYSTEM_ITEM_NOT_DELETABLE	2329	X'00000919'
MQRC_CODED_CHAR_SET_ID_ERROR	2330	X'0000091A'
MQRC_MSG_TOKEN_ERROR	2331	X'0000091B'
MQRC_MISSING_WIH	2332	X'0000091C'
MQRC_WIH_ERROR	2333	X'0000091D'
MQRC_RFH_ERROR	2334	X'0000091E'
MQRC_RFH_STRING_ERROR	2335	X'0000091F'
MQRC_RFH_COMMAND_ERROR	2336	X'00000920'
MQRC_RFH_PARM_ERROR	2337	X'00000921'
MQRC_RFH_DUPLICATE_PARM	2338	X'00000922'
MQRC_RFH_PARM_MISSING	2339	X'00000923'

MQRC_CHAR_CONVERSION_ERROR	2340	X'00000924'
MQRC_UCS2_CONVERSION_ERROR	2341	X'00000925'
MQRC_DB2_NOT_AVAILABLE	2342	X'00000926'
MQRC_OBJECT_NOT_UNIQUE	2343	X'00000927'
MQRC_CONN_TAG_NOT_RELEASED	2344	X'00000928'
MQRC_CF_NOT_AVAILABLE	2345	X'00000929'
MQRC_CF_STRUC_IN_USE	2346	X'0000092A'
MQRC_CF_STRUC_LIST_HDR_IN_USE	2347	X'0000092B'
MQRC_CF_STRUC_AUTH_FAILED	2348	X'0000092C'
MQRC_CF_STRUC_ERROR	2349	X'0000092D'
MQRC_CONN_TAG_NOT_USABLE	2350	X'0000092E'
MQRC_GLOBAL_UOW_CONFLICT	2351	X'0000092F'
MQRC_LOCAL_UOW_CONFLICT	2352	X'00000930'
MQRC_HANDLE_IN_USE_FOR_UOW	2353	X'00000931'
MQRC_UOW_ENLISTMENT_ERROR	2354	X'00000932'
MQRC_UOW_MIX_NOT_SUPPORTED	2355	X'00000933'
MQRC_WXP_ERROR	2356	X'00000934'
MQRC_CURRENT_RECORD_ERROR	2357	X'00000935'
MQRC_NEXT_OFFSET_ERROR	2358	X'00000936'
MQRC_NO_RECORD_AVAILABLE	2359	X'00000937'
MQRC_OBJECT_LEVEL_INCOMPATIBLE	2360	X'00000938'
MQRC_NEXT_RECORD_ERROR	2361	X'00000939'
MQRC_BACKOUT_THRESHOLD_REACHED	2362	X'0000093A'
MQRC_MSG_NOT_MATCHED	2363	X'0000093B'
MQRC_JMS_FORMAT_ERROR	2364	X'0000093C'
MQRC_SEGMENTS_NOT_SUPPORTED	2365	X'0000093D'
MQRC_WRONG_CF_LEVEL	2366	X'0000093E'
MQRC_CONFIG_CREATE_OBJECT	2367	X'0000093F'
MQRC_CONFIG_CHANGE_OBJECT	2368	X'00000940'
MQRC_CONFIG_DELETE_OBJECT	2369	X'00000941'
MQRC_CONFIG_REFRESH_OBJECT	2370	X'00000942'
MQRC_CHANNEL_SSL_ERROR	2371	X'00000943'
MQRC_PARTICIPANT_NOT_DEFINED	2372	X'00000944'
MQRC_CF_STRUC_FAILED	2373	X'00000945'
MQRC_API_EXIT_ERROR	2374	X'00000946'
MQRC_API_EXIT_INIT_ERROR	2375	X'00000947'
MQRC_API_EXIT_TERM_ERROR	2376	X'00000948'
MQRC_EXIT_REASON_ERROR	2377	X'00000949'
MQRC_RESERVED_VALUE_ERROR	2378	X'0000094A'
MQRC_NO_DATA_AVAILABLE	2379	X'0000094B'
MQRC_SCO_ERROR	2380	X'0000094C'
MQRC_KEY_REPOSITORY_ERROR	2381	X'0000094D'

MQRC_CRYPTO_HARDWARE_ERROR	2382	X'0000094E'
MQRC_AUTH_INFO_REC_COUNT_ERROR	2383	X'0000094F'
MQRC_AUTH_INFO_REC_ERROR	2384	X'00000950'
MQRC_AIR_ERROR	2385	X'00000951'
MQRC_AUTH_INFO_TYPE_ERROR	2386	X'00000952'
MQRC_AUTH_INFO_CONN_NAME_ERROR	2387	X'00000953'
MQRC_LDAP_USER_NAME_ERROR	2388	X'00000954'
MQRC_LDAP_USER_NAME_LENGTH_ERR	2389	X'00000955'
MQRC_LDAP_PASSWORD_ERROR	2390	X'00000956'
MQRC_SSL_ALREADY_INITIALIZED	2391	X'00000957'
MQRC_SSL_CONFIG_ERROR	2392	X'00000958'
MQRC_SSL_INITIALIZATION_ERROR	2393	X'00000959'
MQRC_Q_INDEX_TYPE_ERROR	2394	X'0000095A'
MQRC_CFBS_ERROR	2395	X'0000095B'
MQRC_SSL_NOT_ALLOWED	2396	X'0000095C'
MQRC_JSSE_ERROR	2397	X'0000095D'
MQRC_SSL_PEER_NAME_MISMATCH	2398	X'0000095E'
MQRC_SSL_PEER_NAME_ERROR	2399	X'0000095F'
MQRC_UNSUPPORTED_CIPHER_SUITE	2400	X'00000960'
MQRC_SSL_CERTIFICATE_REVOKED	2401	X'00000961'
MQRC_SSL_CERT_STORE_ERROR	2402	X'00000962'
MQRC_CLIENT_EXIT_LOAD_ERROR	2406	X'00000966'
MQRC_CLIENT_EXIT_ERROR	2407	X'00000967'
MQRC_UOW_COMMITTED	2408	X'00000968'
MQRC_SSL_KEY_RESET_ERROR	2409	X'00000969'
MQRC_UNKNOWN_COMPONENT_NAME	2410	X'0000096A'
MQRC_LOGGER_STATUS	2411	X'0000096B'
MQRC_COMMAND_MQSC	2412	X'0000096C'
MQRC_COMMAND_PCF	2413	X'0000096D'
MQRC_CFIF_ERROR	2414	X'0000096E'
MQRC_CFSF_ERROR	2415	X'0000096F'
MQRC_CFGR_ERROR	2416	X'00000970'
MQRC_MSG_NOT_ALLOWED_IN_GROUP	2417	X'00000971'
MQRC_FILTER_OPERATOR_ERROR	2418	X'00000972'
MQRC_NESTED_SELECTOR_ERROR	2419	X'00000973'
MQRC_EPH_ERROR	2420	X'00000974'
MQRC_RFH_FORMAT_ERROR	2421	X'00000975'
MQRC_CFBF_ERROR	2422	X'00000976'
MQRC_CLIENT_CHANNEL_CONFLICT	2423	X'00000977'
MQRC_SD_ERROR	2424	X'00000978'
MQRC_TOPIC_STRING_ERROR	2425	X'00000979'
MQRC_STS_ERROR	2426	X'0000097A'

MQRC_NO_SUBSCRIPTION	2428	X'0000097C'
MQRC_SUBSCRIPTION_IN_USE	2429	X'0000097D'
MQRC_STAT_TYPE_ERROR	2430	X'0000097E'
MQRC_SUB_USER_DATA_ERROR	2431	X'0000097F'
MQRC_SUB_ALREADY_EXISTS	2432	X'00000980'
MQRC_IDENTITY_MISMATCH	2434	X'00000982'
MQRC_ALTER_SUB_ERROR	2435	X'00000983'
MQRC_DURABILITY_NOT_ALLOWED	2436	X'00000984'
MQRC_NO_RETAINED_MSG	2437	X'00000985'
MQRC_SRO_ERROR	2438	X'00000986'
MQRC_SUB_NAME_ERROR	2440	X'00000988'
MQRC_OBJECT_STRING_ERROR	2441	X'00000989'
MQRC_PROPERTY_NAME_ERROR	2442	X'0000098A'
MQRC_SEGMENTATION_NOT_ALLOWED	2443	X'0000098B'
MQRC_CBD_ERROR	2444	X'0000098C'
MQRC_CTLO_ERROR	2445	X'0000098D'
MQRC_NO_CALLBACKS_ACTIVE	2446	X'0000098E'
MQRC_CALLBACK_NOT_REGISTERED	2448	X'00000990'
MQRC_OPTIONS_CHANGED	2457	X'00000999'
MQRC_READ_AHEAD_MSGS	2458	X'0000099A'
MQRC_SELECTOR_SYNTAX_ERROR	2459	X'0000099B'
MQRC_HMSG_ERROR	2460	X'0000099C'
MQRC_CMHO_ERROR	2461	X'0000099D'
MQRC_DMHO_ERROR	2462	X'0000099E'
MQRC_SMPO_ERROR	2463	X'0000099F'
MQRC_IMPO_ERROR	2464	X'000009A0'
MQRC_PROPERTY_NAME_TOO_BIG	2465	X'000009A1'
MQRC_PROP_VALUE_NOT_CONVERTED	2466	X'000009A2'
MQRC_PROP_TYPE_NOT_SUPPORTED	2467	X'000009A3'
MQRC_PROPERTY_VALUE_TOO_BIG	2469	X'000009A5'
MQRC_PROP_CONV_NOT_SUPPORTED	2470	X'000009A6'
MQRC_PROPERTY_NOT_AVAILABLE	2471	X'000009A7'
MQRC_PROP_NUMBER_FORMAT_ERROR	2472	X'000009A8'
MQRC_PROPERTY_TYPE_ERROR	2473	X'000009A9'
MQRC_PROPERTIES_TOO_BIG	2478	X'000009AE'
MQRC_PUT_NOT_RETAINED	2479	X'000009AF'
MQRC_ALIAS_TARGTYPE_CHANGED	2480	X'000009B0'
MQRC_DMPO_ERROR	2481	X'000009B1'
MQRC_PD_ERROR	2482	X'000009B2'
MQRC_CALLBACK_TYPE_ERROR	2483	X'000009B3'
MQRC_CBD_OPTIONS_ERROR	2484	X'000009B4'
MQRC_MAX_MSG_LENGTH_ERROR	2485	X'000009B5'

MQRC_CALLBACK_ROUTINE_ERROR	2486	X'000009B6'
MQRC_CALLBACK_LINK_ERROR	2487	X'000009B7'
MQRC_OPERATION_ERROR	2488	X'000009B8'
MQRC_BMHO_ERROR	2489	X'000009B9'
MQRC_UNSUPPORTED_PROPERTY	2490	X'000009BA'
MQRC_PROP_NAME_NOT_CONVERTED	2492	X'000009BC'
MQRC_GET_ENABLED	2494	X'000009BE'
MQRC_MODULE_NOT_FOUND	2495	X'000009BF'
MQRC_MODULE_INVALID	2496	X'000009C0'
MQRC_MODULE_ENTRY_NOT_FOUND	2497	X'000009C1'
MQRC_MIXED_CONTENT_NOT_ALLOWED	2498	X'000009C2'
MQRC_MSG_HANDLE_IN_USE	2499	X'000009C3'
MQRC_HCONN_ASYNC_ACTIVE	2500	X'000009C4'
MQRC_MHBO_ERROR	2501	X'000009C5'
MQRC_PUBLICATION_FAILURE	2502	X'000009C6'
MQRC_SUB_INHIBITED	2503	X'000009C7'
MQRC_SELECTOR_ALWAYS_FALSE	2504	X'000009C8'
MQRC_XEPO_ERROR	2507	X'000009CB'
MQRC_DURABILITY_NOT_ALTERABLE	2509	X'000009CD'
MQRC_TOPIC_NOT_ALTERABLE	2510	X'000009CE'
MQRC_SUBLEVEL_NOT_ALTERABLE	2512	X'000009D0'
MQRC_PROPERTY_NAME_LENGTH_ERR	2513	X'000009D1'
MQRC_DUPLICATE_GROUP_SUB	2514	X'000009D2'
MQRC_GROUPING_NOT_ALTERABLE	2515	X'000009D3'
MQRC_SELECTOR_INVALID_FOR_TYPE	2516	X'000009D4'
MQRC_HOBJ_QUIESCED	2517	X'000009D5'
MQRC_HOBJ_QUIESCED_NO_MSGS	2518	X'000009D6'
MQRC_SELECTION_STRING_ERROR	2519	X'000009D7'
MQRC_RES_OBJECT_STRING_ERROR	2520	X'000009D8'
MQRC_CONNECTION_SUSPENDED	2521	X'000009D9'
MQRC_INVALID_DESTINATION	2522	X'000009DA'
MQRC_INVALID_SUBSCRIPTION	2523	X'000009DB'
MQRC_SELECTOR_NOT_ALTERABLE	2524	X'000009DC'
MQRC_RETAINED_MSG_Q_ERROR	2525	X'000009DD'
MQRC_RETAINED_NOT_DELIVERED	2526	X'000009DE'
MQRC_RFH_RESTRICTED_FORMAT_ERR	2527	X'000009DF'
MQRC_CONNECTION_STOPPED	2528	X'000009E0'
MQRC_ASYNC_UOW_CONFLICT	2529	X'000009E1'
MQRC_ASYNC_XA_CONFLICT	2530	X'000009E2'
MQRC_PUBSUB_INHIBITED	2531	X'000009E3'
MQRC_MSG_HANDLE_COPY_FAILURE	2532	X'000009E4'
MQRC_DEST_CLASS_NOT_ALTERABLE	2533	X'000009E5'

MQRC_OPERATION_NOT_ALLOWED	2534	X'000009E6'
MQRC_ACTION_ERROR	2535	X'000009E7'
MQRC_CHANNEL_NOT_AVAILABLE	2537	X'000009E9'
MQRC_HOST_NOT_AVAILABLE	2538	X'000009EA'
MQRC_CHANNEL_CONFIG_ERROR	2539	X'000009EB'
MQRC_UNKNOWN_CHANNEL_NAME	2540	X'000009EC'
MQRC_LOOPING_PUBLICATION	2541	X'000009ED'
MQRC_ALREADY_JOINED	2542	X'000009EE'
MQRC_CHANNEL_SSL_WARNING	2552	X'000009F8'
MQRC_OCSP_URL_ERROR	2553	X'000009F9'
MQRC_CIPHER_SPEC_NOT_SUITE_B	2591	X'00000A1F'
MQRC_SUITE_B_ERROR	2592	X'00000A20'
MQRC_REOPEN_EXCL_INPUT_ERROR	6100	X'000017D4'
MQRC_REOPEN_INQUIRE_ERROR	6101	X'000017D5'
MQRC_REOPEN_SAVED_CONTEXT_ERR	6102	X'000017D6'
MQRC_REOPEN_TEMPORARY_Q_ERROR	6103	X'000017D7'
MQRC_ATTRIBUTE_LOCKED	6104	X'000017D8'
MQRC_CURSOR_NOT_VALID	6105	X'000017D9'
MQRC_ENCODING_ERROR	6106	X'000017DA'
MQRC_STRUC_ID_ERROR	6107	X'000017DB'
MQRC_NULL_POINTER	6108	X'000017DC'
MQRC_NO_CONNECTION_REFERENCE	6109	X'000017DD'
MQRC_NO_BUFFER	6110	X'000017DE'
MQRC_BINARY_DATA_LENGTH_ERROR	6111	X'000017DF'
MQRC_BUFFER_NOT_AUTOMATIC	6112	X'000017E0'
MQRC_INSUFFICIENT_BUFFER	6113	X'000017E1'
MQRC_INSUFFICIENT_DATA	6114	X'000017E2'
MQRC_DATA_TRUNCATED	6115	X'000017E3'
MQRC_ZERO_LENGTH	6116	X'000017E4'
MQRC_NEGATIVE_LENGTH	6117	X'000017E5'
MQRC_NEGATIVE_OFFSET	6118	X'000017E6'
MQRC_INCONSISTENT_FORMAT	6119	X'000017E7'
MQRC_INCONSISTENT_OBJECT_STATE	6120	X'000017E8'
MQRC_CONTEXT_OBJECT_NOT_VALID	6121	X'000017E9'
MQRC_CONTEXT_OPEN_ERROR	6122	X'000017EA'
MQRC_STRUC_LENGTH_ERROR	6123	X'000017EB'
MQRC_NOT_CONNECTED	6124	X'000017EC'
MQRC_NOT_OPEN	6125	X'000017ED'
MQRC_DISTRIBUTION_LIST_EMPTY	6126	X'000017EE'
MQRC_INCONSISTENT_OPEN_OPTIONS	6127	X'000017EF'
MQRC_WRONG_VERSION	6128	X'000017F0'
MQRC_REFERENCE_ERROR	6129	X'000017F1'

MQRCCF_* (Command format header Reason Codes):

MQRCCF_CFH_TYPE_ERROR	3001	X'00000BB9'
MQRCCF_CFH_LENGTH_ERROR	3002	X'00000BBA'
MQRCCF_CFH_VERSION_ERROR	3003	X'00000BBB'
MQRCCF_CFH_MSG_SEQ_NUMBER_ERR	3004	X'00000BBC'
MQRCCF_CFH_CONTROL_ERROR	3005	X'00000BBD'
MQRCCF_CFH_PARM_COUNT_ERROR	3006	X'00000BBE'
MQRCCF_CFH_COMMAND_ERROR	3007	X'00000BBF'
MQRCCF_COMMAND_FAILED	3008	X'00000BC0'
MQRCCF_CFIN_LENGTH_ERROR	3009	X'00000BC1'
MQRCCF_CFST_LENGTH_ERROR	3010	X'00000BC2'
MQRCCF_CFST_STRING_LENGTH_ERR	3011	X'00000BC3'
MQRCCF_FORCE_VALUE_ERROR	3012	X'00000BC4'
MQRCCF_STRUCTURE_TYPE_ERROR	3013	X'00000BC5'
MQRCCF_CFIN_PARM_ID_ERROR	3014	X'00000BC6'
MQRCCF_CFST_PARM_ID_ERROR	3015	X'00000BC7'
MQRCCF_MSG_LENGTH_ERROR	3016	X'00000BC8'
MQRCCF_CFIN_DUPLICATE_PARM	3017	X'00000BC9'
MQRCCF_CFST_DUPLICATE_PARM	3018	X'00000BCA'
MQRCCF_PARM_COUNT_TOO_SMALL	3019	X'00000BCB'
MQRCCF_PARM_COUNT_TOO_BIG	3020	X'00000BCC'
MQRCCF_Q_ALREADY_IN_CELL	3021	X'00000BCD'
MQRCCF_Q_TYPE_ERROR	3022	X'00000BCE'
MQRCCF_MD_FORMAT_ERROR	3023	X'00000BCF'
MQRCCF_CFSL_LENGTH_ERROR	3024	X'00000BD0'
MQRCCF_REPLACE_VALUE_ERROR	3025	X'00000BD1'
MQRCCF_CFIL_DUPLICATE_VALUE	3026	X'00000BD2'
MQRCCF_CFIL_COUNT_ERROR	3027	X'00000BD3'
MQRCCF_CFIL_LENGTH_ERROR	3028	X'00000BD4'
MQRCCF_QUIESCE_VALUE_ERROR	3029	X'00000BD5'
MQRCCF_MODE_VALUE_ERROR	3029	X'00000BD5'
MQRCCF_MSG_SEQ_NUMBER_ERROR	3030	X'00000BD6'
MQRCCF_PING_DATA_COUNT_ERROR	3031	X'00000BD7'
MQRCCF_PING_DATA_COMPARE_ERROR	3032	X'00000BD8'
MQRCCF_CFSL_PARM_ID_ERROR	3033	X'00000BD9'
MQRCCF_CHANNEL_TYPE_ERROR	3034	X'00000BDA'
MQRCCF_PARM_SEQUENCE_ERROR	3035	X'00000BDB'
MQRCCF_XMIT_PROTOCOL_TYPE_ERR	3036	X'00000BDC'
MQRCCF_BATCH_SIZE_ERROR	3037	X'00000BDD'
MQRCCF_DISC_INT_ERROR	3038	X'00000BDE'

MQRCCF_SHORT_RETRY_ERROR	3039	X'00000BDF'
MQRCCF_SHORT_TIMER_ERROR	3040	X'00000BE0'
MQRCCF_LONG_RETRY_ERROR	3041	X'00000BE1'
MQRCCF_LONG_TIMER_ERROR	3042	X'00000BE2'
MQRCCF_SEQ_NUMBER_WRAP_ERROR	3043	X'00000BE3'
MQRCCF_MAX_MSG_LENGTH_ERROR	3044	X'00000BE4'
MQRCCF_PUT_AUTH_ERROR	3045	X'00000BE5'
MQRCCF_PURGE_VALUE_ERROR	3046	X'00000BE6'
MQRCCF_CFIL_PARM_ID_ERROR	3047	X'00000BE7'
MQRCCF_MSG_TRUNCATED	3048	X'00000BE8'
MQRCCF_CCSID_ERROR	3049	X'00000BE9'
MQRCCF_ENCODING_ERROR	3050	X'00000BEA'
MQRCCF_QUEUES_VALUE_ERROR	3051	X'00000BEB'
MQRCCF_DATA_CONV_VALUE_ERROR	3052	X'00000BEC'
MQRCCF_INDOUBT_VALUE_ERROR	3053	X'00000BED'
MQRCCF_ESCAPE_TYPE_ERROR	3054	X'00000BEE'
MQRCCF_REPOS_VALUE_ERROR	3055	X'00000BEF'
MQRCCF_CHANNEL_TABLE_ERROR	3062	X'00000BF6'
MQRCCF_MCA_TYPE_ERROR	3063	X'00000BF7'
MQRCCF_CHL_INST_TYPE_ERROR	3064	X'00000BF8'
MQRCCF_CHL_STATUS_NOT_FOUND	3065	X'00000BF9'
MQRCCF_CFSL_DUPLICATE_PARM	3066	X'00000BFA'
MQRCCF_CFSL_TOTAL_LENGTH_ERROR	3067	X'00000BFB'
MQRCCF_CFSL_COUNT_ERROR	3068	X'00000BFC'
MQRCCF_CFSL_STRING_LENGTH_ERR	3069	X'00000BFD'
MQRCCF_BROKER_DELETED	3070	X'00000BFE'
MQRCCF_STREAM_ERROR	3071	X'00000BFF'
MQRCCF_TOPIC_ERROR	3072	X'00000C00'
MQRCCF_NOT_REGISTERED	3073	X'00000C01'
MQRCCF_Q_MGR_NAME_ERROR	3074	X'00000C02'
MQRCCF_INCORRECT_STREAM	3075	X'00000C03'
MQRCCF_Q_NAME_ERROR	3076	X'00000C04'
MQRCCF_NO_RETAINED_MSG	3077	X'00000C05'
MQRCCF_DUPLICATE_IDENTITY	3078	X'00000C06'
MQRCCF_INCORRECT_Q	3079	X'00000C07'
MQRCCF_CORREL_ID_ERROR	3080	X'00000C08'
MQRCCF_NOT_AUTHORIZED	3081	X'00000C09'
MQRCCF_UNKNOWN_STREAM	3082	X'00000C0A'
MQRCCF_REG_OPTIONS_ERROR	3083	X'00000C0B'
MQRCCF_PUB_OPTIONS_ERROR	3084	X'00000C0C'
MQRCCF_UNKNOWN_BROKER	3085	X'00000C0D'
MQRCCF_Q_MGR_CCSID_ERROR	3086	X'00000C0E'

MQRCCF_DEL_OPTIONS_ERROR	3087	X'00000C0F'
MQRCCF_CLUSTER_NAME_CONFLICT	3088	X'00000C10'
MQRCCF_REPOS_NAME_CONFLICT	3089	X'00000C11'
MQRCCF_CLUSTER_Q_USAGE_ERROR	3090	X'00000C12'
MQRCCF_ACTION_VALUE_ERROR	3091	X'00000C13'
MQRCCF_COMMS_LIBRARY_ERROR	3092	X'00000C14'
MQRCCF_NETBIOS_NAME_ERROR	3093	X'00000C15'
MQRCCF_BROKER_COMMAND_FAILED	3094	X'00000C16'
MQRCCF_CFST_CONFLICTING_PARM	3095	X'00000C17'
MQRCCF_PATH_NOT_VALID	3096	X'00000C18'
MQRCCF_PARM_SYNTAX_ERROR	3097	X'00000C19'
MQRCCF_PWD_LENGTH_ERROR	3098	X'00000C1A'
MQRCCF_FILTER_ERROR	3150	X'00000C4E'
MQRCCF_WRONG_USER	3151	X'00000C4F'
MQRCCF_DUPLICATE_SUBSCRIPTION	3152	X'00000C50'
MQRCCF_SUB_NAME_ERROR	3153	X'00000C51'
MQRCCF_SUB_IDENTITY_ERROR	3154	X'00000C52'
MQRCCF_SUBSCRIPTION_IN_USE	3155	X'00000C53'
MQRCCF_SUBSCRIPTION_LOCKED	3156	X'00000C54'
MQRCCF_ALREADY_JOINED	3157	X'00000C55'
MQRCCF_OBJECT_IN_USE	3160	X'00000C58'
MQRCCF_UNKNOWN_FILE_NAME	3161	X'00000C59'
MQRCCF_FILE_NOT_AVAILABLE	3162	X'00000C5A'
MQRCCF_DISC_RETRY_ERROR	3163	X'00000C5B'
MQRCCF_ALLOC_RETRY_ERROR	3164	X'00000C5C'
MQRCCF_ALLOC_SLOW_TIMER_ERROR	3165	X'00000C5D'
MQRCCF_ALLOC_FAST_TIMER_ERROR	3166	X'00000C5E'
MQRCCF_PORT_NUMBER_ERROR	3167	X'00000C5F'
MQRCCF_CHL_SYSTEM_NOT_ACTIVE	3168	X'00000C60'
MQRCCF_ENTITY_NAME_MISSING	3169	X'00000C61'
MQRCCF_PROFILE_NAME_ERROR	3170	X'00000C62'
MQRCCF_AUTH_VALUE_ERROR	3171	X'00000C63'
MQRCCF_AUTH_VALUE_MISSING	3172	X'00000C64'
MQRCCF_OBJECT_TYPE_MISSING	3173	X'00000C65'
MQRCCF_CONNECTION_ID_ERROR	3174	X'00000C66'
MQRCCF_LOG_TYPE_ERROR	3175	X'00000C67'
MQRCCF_PROGRAM_NOT_AVAILABLE	3176	X'00000C68'
MQRCCF_PROGRAM_AUTH_FAILED	3177	X'00000C69'
MQRCCF_NONE_FOUND	3200	X'00000C80'
MQRCCF_SECURITY_SWITCH_OFF	3201	X'00000C81'
MQRCCF_SECURITY_REFRESH_FAILED	3202	X'00000C82'
MQRCCF_PARM_CONFLICT	3203	X'00000C83'

MQRCCF_COMMAND_INHIBITED	3204	X'00000C84'
MQRCCF_OBJECT_BEING_DELETED	3205	X'00000C85'
MQRCCF_STORAGE_CLASS_IN_USE	3207	X'00000C87'
MQRCCF_OBJECT_NAME_RESTRICTED	3208	X'00000C88'
MQRCCF_OBJECT_LIMIT_EXCEEDED	3209	X'00000C89'
MQRCCF_OBJECT_OPEN_FORCE	3210	X'00000C8A'
MQRCCF_DISPOSITION_CONFLICT	3211	X'00000C8B'
MQRCCF_Q_MGR_NOT_IN_QSG	3212	X'00000C8C'
MQRCCF_ATTR_VALUE_FIXED	3213	X'00000C8D'
MQRCCF_NAMELIST_ERROR	3215	X'00000C8F'
MQRCCF_NO_CHANNEL_INITIATOR	3217	X'00000C91'
MQRCCF_CHANNEL_INITIATOR_ERROR	3218	X'00000C92'
MQRCCF_COMMAND_LEVEL_CONFLICT	3222	X'00000C96'
MQRCCF_Q_ATTR_CONFLICT	3223	X'00000C97'
MQRCCF_EVENTS_DISABLED	3224	X'00000C98'
MQRCCF_COMMAND_SCOPE_ERROR	3225	X'00000C99'
MQRCCF_COMMAND_REPLY_ERROR	3226	X'00000C9A'
MQRCCF_FUNCTION_RESTRICTED	3227	X'00000C9B'
MQRCCF_PARM_MISSING	3228	X'00000C9C'
MQRCCF_PARM_VALUE_ERROR	3229	X'00000C9D'
MQRCCF_COMMAND_LENGTH_ERROR	3230	X'00000C9E'
MQRCCF_COMMAND_ORIGIN_ERROR	3231	X'00000C9F'
MQRCCF_LISTENER_CONFLICT	3232	X'00000CA0'
MQRCCF_LISTENER_STARTED	3233	X'00000CA1'
MQRCCF_LISTENER_STOPPED	3234	X'00000CA2'
MQRCCF_CHANNEL_ERROR	3235	X'00000CA3'
MQRCCF_CF_STRUC_ERROR	3236	X'00000CA4'
MQRCCF_UNKNOWN_USER_ID	3237	X'00000CA5'
MQRCCF_UNEXPECTED_ERROR	3238	X'00000CA6'
MQRCCF_NO_XCF_PARTNER	3239	X'00000CA7'
MQRCCF_CFGR_PARM_ID_ERROR	3240	X'00000CA8'
MQRCCF_CFIF_LENGTH_ERROR	3241	X'00000CA9'
MQRCCF_CFIF_OPERATOR_ERROR	3242	X'00000CAA'
MQRCCF_CFIF_PARM_ID_ERROR	3243	X'00000CAB'
MQRCCF_CFSF_FILTER_VAL_LEN_ERR	3244	X'00000CAC'
MQRCCF_CFSF_LENGTH_ERROR	3245	X'00000CAD'
MQRCCF_CFSF_OPERATOR_ERROR	3246	X'00000CAE'
MQRCCF_CFSF_PARM_ID_ERROR	3247	X'00000CAF'
MQRCCF_TOO_MANY_FILTERS	3248	X'00000CB0'
MQRCCF_LISTENER_RUNNING	3249	X'00000CB1'
MQRCCF_LSTR_STATUS_NOT_FOUND	3250	X'00000CB2'
MQRCCF_SERVICE_RUNNING	3251	X'00000CB3'

MQRCCF_SERV_STATUS_NOT_FOUND	3252	X'00000CB4'
MQRCCF_SERVICE_STOPPED	3253	X'00000CB5'
MQRCCF_CFBS_DUPLICATE_PARM	3254	X'00000CB6'
MQRCCF_CFBS_LENGTH_ERROR	3255	X'00000CB7'
MQRCCF_CFBS_PARM_ID_ERROR	3256	X'00000CB8'
MQRCCF_CFBS_STRING_LENGTH_ERR	3257	X'00000CB9'
MQRCCF_CFGR_LENGTH_ERROR	3258	X'00000CBA'
MQRCCF_CFGR_PARM_COUNT_ERROR	3259	X'00000CBB'
MQRCCF_CONN_NOT_STOPPED	3260	X'00000CBC'
MQRCCF_SERVICE_REQUEST_PENDING	3261	X'00000CBD'
MQRCCF_NO_START_CMD	3262	X'00000CBE'
MQRCCF_NO_STOP_CMD	3263	X'00000CBF'
MQRCCF_CFBF_LENGTH_ERROR	3264	X'00000CC0'
MQRCCF_CFBF_PARM_ID_ERROR	3265	X'00000CC1'
MQRCCF_CFBF_OPERATOR_ERROR	3266	X'00000CC2'
MQRCCF_CFBF_FILTER_VAL_LEN_ERR	3267	X'00000CC3'
MQRCCF_LISTENER_STILL_ACTIVE	3268	X'00000CC4'
MQRCCF_DEF_XMIT_Q_CLUS_ERROR	3269	X'0000CC5'
MQRCCF_TOPICSTR_ALREADY_EXISTS	3300	X'00000CE4'
MQRCCF_SHARING_CONVS_ERROR	3301	X'00000CE5'
MQRCCF_SHARING_CONVS_TYPE	3302	X'00000CE6'
MQRCCF_SECURITY_CASE_CONFLICT	3303	X'00000CE7'
MQRCCF_TOPIC_TYPE_ERROR	3305	X'00000CE9'
MQRCCF_MAX_INSTANCES_ERROR	3306	X'00000CEA'
MQRCCF_MAX_INSTS_PER_CLNT_ERR	3307	X'00000CEB'
MQRCCF_TOPIC_STRING_NOT_FOUND	3308	X'00000CEC'
MQRCCF_SUBSCRIPTION_POINT_ERR	3309	X'00000CED'
MQRCCF_SUB_ALREADY_EXISTS	3311	X'00000CEF'
MQRCCF_UNKNOWN_OBJECT_NAME	3312	X'00000CF0'
MQRCCF_REMOTE_Q_NAME_ERROR	3313	X'00000CF1'
MQRCCF_DURABILITY_NOT_ALLOWED	3314	X'00000CF2'
MQRCCF_HOBJ_ERROR	3315	X'00000CF3'
MQRCCF_DEST_NAME_ERROR	3316	X'00000CF4'
MQRCCF_INVALID_DESTINATION	3317	X'00000CF5'
MQRCCF_PUBSUB_INHIBITED	3318	X'00000CF6'
MQRCCF_CHLAUTH_TYPE_ERROR	3326	X'00000CFE'
MQRCCF_CHLAUTH_ACTION_ERROR	3327	X'00000CFF'
MQRCCF_CHLAUTH_USERSRC_ERROR	3335	X'00000D07'
MQRCCF_WRONG_CHLAUTH_TYPE	3336	X'00000D08'
MQRCCF_CHLAUTH_ALREADY_EXISTS	3337	X'00000D09'
MQRCCF_CHLAUTH_NOT_FOUND	3338	X'00000D0A'
MQRCCF_WRONG_CHLAUTH_ACTION	3339	X'00000D0B'

MQRCCF_WRONG_CHLAUTH_USERSRC	3340	X'00000D0C'
MQRCCF_CHLAUTH_WARN_ERROR	3341	X'00000D0D'
MQRCCF_WRONG_CHLAUTH_MATCH	3342	X'00000D0E'
MQRCCF_IPADDR_RANGE_CONFLICT	3343	X'00000D0F'
MQRCCF_CHLAUTH_MAX_EXCEEDED	3344	X'00000D10'
MQRCCF_IPADDR_ERROR	3345	X'00000D11'
MQRCCF_IPADDR_RANGE_ERROR	3346	X'00000D12'
MQRCCF_PROFILE_NAME_MISSING	3347	X'00000D13'
MQRCCF_CHLAUTH_CLNTUSER_ERROR	3348	X'00000D14'
MQRCCF_CHLAUTH_NAME_ERROR	3349	X'00000D15'
MQRCCF_SUITE_B_ERROR	3353	X'00000D19'
MQRCCF_PSCLUS_DISABLED_TOPDEF	3359	X'00000D1F'
MQRCCF_PSCLUS_TOPIC_EXISTS	3360	X'00000D20'
MQRCCF_OBJECT_ALREADY_EXISTS	4001	X'00000FA1'
MQRCCF_OBJECT_WRONG_TYPE	4002	X'00000FA2'
MQRCCF_LIKE_OBJECT_WRONG_TYPE	4003	X'00000FA3'
MQRCCF_OBJECT_OPEN	4004	X'00000FA4'
MQRCCF_ATTR_VALUE_ERROR	4005	X'00000FA5'
MQRCCF_UNKNOWN_Q_MGR	4006	X'00000FA6'
MQRCCF_Q_WRONG_TYPE	4007	X'00000FA7'
MQRCCF_OBJECT_NAME_ERROR	4008	X'00000FA8'
MQRCCF_ALLOCATE_FAILED	4009	X'00000FA9'
MQRCCF_HOST_NOT_AVAILABLE	4010	X'00000FAA'
MQRCCF_CONFIGURATION_ERROR	4011	X'00000FAB'
MQRCCF_CONNECTION_REFUSED	4012	X'00000FAC'
MQRCCF_ENTRY_ERROR	4013	X'00000FAD'
MQRCCF_SEND_FAILED	4014	X'00000FAE'
MQRCCF_RECEIVED_DATA_ERROR	4015	X'00000FAF'
MQRCCF_RECEIVE_FAILED	4016	X'00000FB0'
MQRCCF_CONNECTION_CLOSED	4017	X'00000FB1'
MQRCCF_NO_STORAGE	4018	X'00000FB2'
MQRCCF_NO_COMMS_MANAGER	4019	X'00000FB3'
MQRCCF_LISTENER_NOT_STARTED	4020	X'00000FB4'
MQRCCF_BIND_FAILED	4024	X'00000FB8'
MQRCCF_CHANNEL_INDOUBT	4025	X'00000FB9'
MQRCCF_MQCONN_FAILED	4026	X'00000FBA'
MQRCCF_MQOPEN_FAILED	4027	X'00000FBB'
MQRCCF_MQGET_FAILED	4028	X'00000FBC'
MQRCCF_MQPUT_FAILED	4029	X'00000FBD'
MQRCCF_PING_ERROR	4030	X'00000FBE'
MQRCCF_CHANNEL_IN_USE	4031	X'00000FBF'
MQRCCF_CHANNEL_NOT_FOUND	4032	X'00000FC0'

MQRCCF_UNKNOWN_REMOTE_CHANNEL	4033	X'00000FC1'
MQRCCF_REMOTE_QM_UNAVAILABLE	4034	X'00000FC2'
MQRCCF_REMOTE_QM_TERMINATING	4035	X'00000FC3'
MQRCCF_MQINQ_FAILED	4036	X'00000FC4'
MQRCCF_NOT_XMIT_Q	4037	X'00000FC5'
MQRCCF_CHANNEL_DISABLED	4038	X'00000FC6'
MQRCCF_USER_EXIT_NOT_AVAILABLE	4039	X'00000FC7'
MQRCCF_COMMIT_FAILED	4040	X'00000FC8'
MQRCCF_WRONG_CHANNEL_TYPE	4041	X'00000FC9'
MQRCCF_CHANNEL_ALREADY_EXISTS	4042	X'00000FCA'
MQRCCF_DATA_TOO_LARGE	4043	X'00000FCB'
MQRCCF_CHANNEL_NAME_ERROR	4044	X'00000FCC'
MQRCCF_XMIT_Q_NAME_ERROR	4045	X'00000FCD'
MQRCCF_MCA_NAME_ERROR	4047	X'00000FCF'
MQRCCF_SEND_EXIT_NAME_ERROR	4048	X'00000FD0'
MQRCCF_SEC_EXIT_NAME_ERROR	4049	X'00000FD1'
MQRCCF_MSG_EXIT_NAME_ERROR	4050	X'00000FD2'
MQRCCF_RCV_EXIT_NAME_ERROR	4051	X'00000FD3'
MQRCCF_XMIT_Q_NAME_WRONG_TYPE	4052	X'00000FD4'
MQRCCF_MCA_NAME_WRONG_TYPE	4053	X'00000FD5'
MQRCCF_DISC_INT_WRONG_TYPE	4054	X'00000FD6'
MQRCCF_SHORT_RETRY_WRONG_TYPE	4055	X'00000FD7'
MQRCCF_SHORT_TIMER_WRONG_TYPE	4056	X'00000FD8'
MQRCCF_LONG_RETRY_WRONG_TYPE	4057	X'00000FD9'
MQRCCF_LONG_TIMER_WRONG_TYPE	4058	X'00000FDA'
MQRCCF_PUT_AUTH_WRONG_TYPE	4059	X'00000FDB'
MQRCCF_KEEP_ALIVE_INT_ERROR	4060	X'00000FDC'
MQRCCF_MISSING_CONN_NAME	4061	X'00000FDD'
MQRCCF_CONN_NAME_ERROR	4062	X'00000FDE'
MQRCCF_MQSET_FAILED	4063	X'00000FDF'
MQRCCF_CHANNEL_NOT_ACTIVE	4064	X'00000FE0'
MQRCCF_TERMINATED_BY_SEC_EXIT	4065	X'00000FE1'
MQRCCF_DYNAMIC_Q_SCOPE_ERROR	4067	X'00000FE3'
MQRCCF_CELL_DIR_NOT_AVAILABLE	4068	X'00000FE4'
MQRCCF_MR_COUNT_ERROR	4069	X'00000FE5'
MQRCCF_MR_COUNT_WRONG_TYPE	4070	X'00000FE6'
MQRCCF_MR_EXIT_NAME_ERROR	4071	X'00000FE7'
MQRCCF_MR_EXIT_NAME_WRONG_TYPE	4072	X'00000FE8'
MQRCCF_MR_INTERVAL_ERROR	4073	X'00000FE9'
MQRCCF_MR_INTERVAL_WRONG_TYPE	4074	X'00000FEA'
MQRCCF_NPM_SPEED_ERROR	4075	X'00000FEB'
MQRCCF_NPM_SPEED_WRONG_TYPE	4076	X'00000FEC'

MQRCCF_HB_INTERVAL_ERROR	4077	X'00000FED'
MQRCCF_HB_INTERVAL_WRONG_TYPE	4078	X'00000FEE'
MQRCCF_CHAD_ERROR	4079	X'00000FEF'
MQRCCF_CHAD_WRONG_TYPE	4080	X'00000FF0'
MQRCCF_CHAD_EVENT_ERROR	4081	X'00000FF1'
MQRCCF_CHAD_EVENT_WRONG_TYPE	4082	X'00000FF2'
MQRCCF_CHAD_EXIT_ERROR	4083	X'00000FF3'
MQRCCF_CHAD_EXIT_WRONG_TYPE	4084	X'00000FF4'
MQRCCF_SUPPRESSED_BY_EXIT	4085	X'00000FF5'
MQRCCF_BATCH_INT_ERROR	4086	X'00000FF6'
MQRCCF_BATCH_INT_WRONG_TYPE	4087	X'00000FF7'
MQRCCF_NET_PRIORITY_ERROR	4088	X'00000FF8'
MQRCCF_NET_PRIORITY_WRONG_TYPE	4089	X'00000FF9'
MQRCCF_CHANNEL_CLOSED	4090	X'00000FFA'
MQRCCF_Q_STATUS_NOT_FOUND	4091	X'00000FFB'
MQRCCF_SSL_CIPHER_SPEC_ERROR	4092	X'00000FFC'
MQRCCF_SSL_PEER_NAME_ERROR	4093	X'00000FFD'
MQRCCF_SSL_CLIENT_AUTH_ERROR	4094	X'00000FFE'
MQRCCF_RETAINED_NOT_SUPPORTED	4095	X'00000FFF'

MQRCN_* (Client reconnect Constants):

MQRCN_NO	0	X'00000000'
MQRCN_YES	1	X'00000001'
MQRCN_Q_MGR	2	X'00000002'
MQRCN_DISABLED	3	X'00000003'

MQRCVTIME_* (Receive Timeout Types):

MQRCVTIME_MULTIPLY	0	X'00000000'
MQRCVTIME_ADD	1	X'00000001'
MQRCVTIME_EQUAL	2	X'00000002'

MQREADA_* (Read Ahead Values):

MQREADA_NO	0	X'00000000'
MQREADA_YES	1	X'00000001'
MQREADA_DISABLED	2	X'00000002'
MQREADA_INHIBITED	3	X'00000003'
MQREADA_BACKLOG	4	X'00000004'

MQRECORDING_* (Recording Options):

MQRECORDING_DISABLED	0	X'00000000'
MQRECORDING_Q	1	X'00000001'
MQRECORDING_MSG	2	X'00000002'

MQREGO_* (Publish/Subscribe Registration Options):

MQREGO_NONE	0	X'00000000'
MQREGO_CORREL_ID_AS_IDENTITY	1	X'00000001'
MQREGO_ANONYMOUS	2	X'00000002'
MQREGO_LOCAL	4	X'00000004'
MQREGO_DIRECT_REQUESTS	8	X'00000008'
MQREGO_NEW_PUBLICATIONS_ONLY	16	X'00000010'
MQREGO_PUBLISH_ON_REQUEST_ONLY	32	X'00000020'
MQREGO_DEREGISTER_ALL	64	X'00000040'
MQREGO_INCLUDE_STREAM_NAME	128	X'00000080'
MQREGO_INFORM_IF_RETAINED	256	X'00000100'
MQREGO_DUPLICATES_OK	512	X'00000200'
MQREGO_NON_PERSISTENT	1024	X'00000400'
MQREGO_PERSISTENT	2048	X'00000800'
MQREGO_PERSISTENT_AS_PUBLISH	4096	X'00001000'
MQREGO_PERSISTENT_AS_Q	8192	X'00002000'
MQREGO_ADD_NAME	16384	X'00004000'
MQREGO_NO_ALTERATION	32768	X'00008000'
MQREGO_FULL_RESPONSE	65536	X'00010000'
MQREGO_JOIN_SHARED	131072	X'00020000'
MQREGO_JOIN_EXCLUSIVE	262144	X'00040000'
MQREGO_LEAVE_ONLY	524288	X'00080000'
MQREGO_VARIABLE_USER_ID	1048576	X'00100000'
MQREGO_LOCKED	2097152	X'00200000'

MQRFH_* (Rules and formatting header structure and Flags):
Rules and formatting header structure

MQRFH_STRUC_ID	"RFHb"	
MQRFH_STRUC_ID_ARRAY	'R','F','H','b'	
MQRFH_VERSION_1	1	X'00000001'
MQRFH_VERSION_2	2	X'00000002'
MQRFH_STRUC_LENGTH_FIXED	32	X'00000020'
MQRFH_STRUC_LENGTH_FIXED_2	36	X'00000024'

Rules and formatting header Flags

MQRFH_NONE	0	X'00000000'
MQRFH_NO_FLAGS	0	X'00000000'

MQRFH2_* (Publish/Subscribe Options Tag RFH2 Top-level folder Tags):

MQRFH2_NAME_VALUE_VERSION	1	X'00000001'

MQRFH2_* (Publish/Subscribe Options Tag Tag names):

MQRFH2_PUBSUB_CMD_FOLDER	"psc"	
MQRFH2_PUBSUB_RESP_FOLDER	"pscr"	
MQRFH2_MSG_CONTENT_FOLDER	"mcd"	
MQRFH2_USER_FOLDER	"usr"	

MQRFH2_* (Publish/Subscribe Options Tag XML tag names):

MQRFH2_PUBSUB_CMD_FOLDER_B	"<psc>"	
MQRFH2_PUBSUB_CMD_FOLDER_E	"</psc>"	
MQRFH2_PUBSUB_RESP_FOLDER_B	"<pscr>"	
MQRFH2_PUBSUB_RESP_FOLDER_E	"</pscr>"	
MQRFH2_MSG_CONTENT_FOLDER_B	"<mcd>"	
MQRFH2_MSG_CONTENT_FOLDER_E	"</mcd>"	
MQRFH2_USER_FOLDER_B	"<usr>"	
MQRFH2_USER_FOLDER_E	"</usr>"	

MQRL_* (Returned Length):

MQRL_UNDEFINED	-1	X'FFFFFFFF'

MQRMH_* (Reference message header structure):

MQRMH_STRUC_ID	"RMHb"	
MQRMH_STRUC_ID_ARRAY	'R','M','H','b'	
MQRMH_VERSION_1	1	X'00000001'
MQRMH_CURRENT_VERSION	1	X'00000001'

MQRMHF_* (Reference message header Flags):

MQRMHF_LAST	1	X'00000001'
MQRMHF_NOT_LAST	0	X'00000000'

MQRO_* (Report Options):

MQRO_EXCEPTION	16777216	X'01000000'
MQRO_EXCEPTION_WITH_DATA	50331648	X'03000000'
MQRO_EXCEPTION_WITH_FULL_DATA	117440512	X'07000000'
MQRO_EXPIRATION	2097152	X'00200000'
MQRO_EXPIRATION_WITH_DATA	6291456	X'00600000'
MQRO_EXPIRATION_WITH_FULL_DATA	14680064	X'00E00000'
MQRO_COA	256	X'00000100'
MQRO_COA_WITH_DATA	768	X'00000300'
MQRO_COA_WITH_FULL_DATA	1792	X'00000700'
MQRO_COD	2048	X'00000800'
MQRO_COD_WITH_DATA	6144	X'00001800'
MQRO_COD_WITH_FULL_DATA	14336	X'00003800'
MQRO_PAN	1	X'00000001'
MQRO_NAN	2	X'00000002'
MQRO_ACTIVITY	4	X'00000004'
MQRO_NEW_MSG_ID	0	X'00000000'
MQRO_PASS_MSG_ID	128	X'00000080'
MQRO_COPY_MSG_ID_TO_CORREL_ID	0	X'00000000'
MQRO_PASS_CORREL_ID	64	X'00000040'
MQRO_DEAD_LETTER_Q	0	X'00000000'
MQRO_DISCARD_MSG	134217728	X'08000000'
MQRO_PASS_DISCARD_AND_EXPIRY	16384	X'00004000'
MQRO_NONE	0	X'00000000'

MQRO_* (Report Options Masks):

MQRO_REJECT_UNSUP_MASK	270270464	X'101C0000'
MQRO_ACCEPT_UNSUP_MASK	-270532353	X'EFE000FF'
MQRO_ACCEPT_UNSUP_IF_XMIT_MASK	261888	X'0003FF00'

MQROUTE_* (Trace-route):
Trace-route Max Activities (MQIACF_MAX_ACTIVITIES)

MQROUTE_UNLIMITED_ACTIVITIES	0	X'00000000'

Trace-route Detail (MQIACF_ROUTE_DETAIL)

MQROUTE_DETAIL_LOW	2	X'00000002'
MQROUTE_DETAIL_MEDIUM	8	X'00000008'
MQROUTE_DETAIL_HIGH	32	X'00000020'

Trace-route Forwarding (MQIACF_ROUTE_FORWARDING)

MQROUTE_FORWARD_ALL	256	X'00000100'
MQROUTE_FORWARD_IF_SUPPORTED	512	X'00000200'
MQROUTE_FORWARD_REJ_UNSUP_MASK	-65536	X'FFFF0000'

Trace-route Delivery (MQIACF_ROUTE_DELIVERY)

MQROUTE_DELIVER_YES	4096	X'00001000'
MQROUTE_DELIVER_NO	8192	X'00002000'
MQROUTE_DELIVER_REJ_UNSUP_MASK	-65536	X'FFFF0000'

Trace-route Accumulation (MQIACF_ROUTE_ACCUMULATION)

MQROUTE_ACCUMULATE_NONE	65539	X'00010003'
MQROUTE_ACCUMULATE_IN_MSG	65540	X'00010004'
MQROUTE_ACCUMULATE_AND_REPLY	65541	X'00010005'

MQRP_* (Command format Replace Options):

MQRP_YES	1	X'00000001'
MQRP_NO	0	X'00000000'

MQRQ_* (Command format Reason Qualifiers):

MQRQ_CONN_NOT_AUTHORIZED	1	X'00000001'
MQRQ_OPEN_NOT_AUTHORIZED	2	X'00000002'
MQRQ_CLOSE_NOT_AUTHORIZED	3	X'00000003'
MQRQ_CMD_NOT_AUTHORIZED	4	X'00000004'
MQRQ_Q_MGR_STOPPING	5	X'00000005'
MQRQ_Q_MGR_QUIESCING	6	X'00000006'
MQRQ_CHANNEL_STOPPED_OK	7	X'00000007'
MQRQ_CHANNEL_STOPPED_ERROR	8	X'00000008'
MQRQ_CHANNEL_STOPPED_RETRY	9	X'00000009'
MQRQ_CHANNEL_STOPPED_DISABLED	10	X'0000000A'
MQRQ_BRIDGE_STOPPED_OK	11	X'0000000B'
MQRQ_BRIDGE_STOPPED_ERROR	12	X'0000000C'

MQRQ_SSL_HANDSHAKE_ERROR	13	X'0000000D'
MQRQ_SSL_CIPHER_SPEC_ERROR	14	X'0000000E'
MQRQ_SSL_CLIENT_AUTH_ERROR	15	X'0000000F'
MQRQ_SSL_PEER_NAME_ERROR	16	X'00000010'
MQRQ_SUB_NOT_AUTHORIZED	17	X'00000011'
MQRQ_SUB_DEST_NOT_AUTHORIZED	18	X'00000012'
MQRQ_SSL_UNKNOWN_REVOCATION	19	X'00000013'

MQRT_* (Command format Refresh Types):

MQRT_CONFIGURATION	1	X'00000001'
MQRT_EXPIRY	2	X'00000002'
MQRT_NSPROC	3	X'00000003'
MQRT_PROXYSUB	4	X'00000004'

MQRU_* (Request Only):

MQRU_PUBLISH_ON_REQUEST	1	X'00000001'
MQRU_PUBLISH_ALL	2	X'00000002'

MQSCA_* (SSL Client Authentication):

MQSCA_REQUIRED	0	X'00000000'
MQSCA_OPTIONAL	1	X'00000001'

MQSCO_* (SSL configuration options):
SSL configuration options structure

MQSCO_STRUC_ID	"SCOb"	
MQSCO_STRUC_ID_ARRAY	'S','C','O','b'	
MQSCO_VERSION_1	1	X'00000001'
MQSCO_VERSION_2	2	X'00000002'
MQSCO_VERSION_3	3	X'00000003'
MQSCO_VERSION_4	4	X'00000004'
MQSCO_CURRENT_VERSION	4	X'00000004'

SSL configuration options Key Reset Count

MQSCO_RESET_COUNT_DEFAULT	0	X'00000000'

Command format Queue Definition Scope

MQSCO_Q_MGR	1	X'00000001'
MQSCO_CELL	2	X'00000002'

MQSCOPE_* (Publish scope):

MQSCOPE_ALL	0	X'00000000'
MQSCOPE_AS_PARENT	1	X'00000001'
MQSCOPE_QMGR	4	X'00000004'

MQSCYC_* (Security Case):

MQSCYC_UPPER	0	X'00000000'
MQSCYC_MIXED	1	X'00000001'

MQSD_* (Object descriptor structure):

MQSD_STRUC_ID	"SDbb"	
MQSD_STRUC_ID_ARRAY	'S','D','b','b'	
MQSD_VERSION_1	1	X'00000001'
MQSD_CURRENT_VERSION	1	X'00000001'

MQSECITEM_* (Command format Security Items):

MQSECITEM_ALL	0	X'00000000'
MQSECITEM_MQADMIN	1	X'00000001'
MQSECITEM_MQNLIST	2	X'00000002'
MQSECITEM_MQPROC	3	X'00000003'
MQSECITEM_MQQUEUE	4	X'00000004'
MQSECITEM_MQCONN	5	X'00000005'
MQSECITEM_MQCMDS	6	X'00000006'
MQSECITEM_MXADMIN	7	X'00000007'
MQSECITEM_MXNLIST	8	X'00000008'
MQSECITEM_MXPROC	9	X'00000009'
MQSECITEM_MXQUEUE	10	X'0000000A'
MQSECITEM_MXTOPIC	11	X'0000000B'

MQSECSW_* (Command format Security Switches and Switch States):
Command format Security Switches

MQSECSW_PROCESS	1	X'00000001'
MQSECSW_NAMELIST	2	X'00000002'
MQSECSW_Q	3	X'00000003'
MQSECSW_TOPIC	4	X'00000004'
MQSECSW_CONTEXT	6	X'00000006'
MQSECSW_ALTERNATE_USER	7	X'00000007'
MQSECSW_COMMAND	8	X'00000008'
MQSECSW_CONNECTION	9	X'00000009'
MQSECSW_SUBSYSTEM	10	X'0000000A'
MQSECSW_COMMAND_RESOURCES	11	X'0000000B'
MQSECSW_Q_MGR	15	X'0000000F'
MQSECSW_QSG	16	X'00000010'

Command format Security Switch States

MQSECSW_OFF_FOUND	21	X'00000015'
MQSECSW_ON_FOUND	22	X'00000016'
MQSECSW_OFF_NOT_FOUND	23	X'00000017'
MQSECSW_ON_NOT_FOUND	24	X'00000018'
MQSECSW_OFF_ERROR	25	X'00000019'
MQSECSW_ON_OVERRIDDEN	26	X'0000001A'

MQSECTYPE_* (Command format Security Types):

MQSECTYPE_AUTHSERV	1	X'00000001'
MQSECTYPE_SSL	2	X'00000002'
MQSECTYPE_CLASSES	3	X'00000003'

MQSEG_* (Segmentation):

MQSEG_INHIBITED	'b'	
MQSEG_ALLOWED	'A'	

MQSEL_* (Special Selector Values):

MQSEL_ANY_SELECTOR	-30001	X'FFFF8ACF'
MQSEL_ANY_USER_SELECTOR	-30002	X'FFFF8ACE'
MQSEL_ANY_SYSTEM_SELECTOR	-30003	X'FFFF8ACD'
MQSEL_ALL_SELECTORS	-30001	X'FFFF8ACF'
MQSEL_ALL_USER_SELECTORS	-30002	X'FFFF8ACE'
MQSEL_ALL_SYSTEM_SELECTORS	-30003	X'FFFF8ACD'

MQSELTYPE_* (Selector Types):

MQSELTYPE_NONE	0	X'00000000'
MQSELTYPE_STANDARD	1	X'00000001'
MQSELTYPE_EXTENDED	2	X'00000002'

MQSID_* (Security Identifier):

MQSID_NONE	X'00...00'	(40 nulls)
MQSID_NONE_ARRAY	'\0','\0',...	(40 nulls)

MQSIDT_* (Security Identifier Types):

MQSIDT_NONE	X'00'	
MQSIDT_NT_SECURITY_ID	X'01'	
MQSIDT_WAS_SECURITY_ID	X'02'	

MQSMPO_* (Set message property options and structure):
Set message property options structure

MQSMPO_STRUC_ID	"SMPO"	
MQSMPO_STRUC_ID_ARRAY	'S','M','P','O'	
MQSMPO_VERSION_1	1	X'00000001'
MQSMPO_CURRENT_VERSION	1	X'00000001'

Set Message Property Options

MQSMPO_SET_FIRST	0	X'00000000'
MQSMPO_SET_PROP_UNDER_CURSOR	1	X'00000001'
MQSMPO_SET_PROP_AFTER_CURSOR	2	X'00000002'
MQSMPO_APPEND_PROPERTY	4	X'00000004'
MQSMPO_SET_PROP_BEFORE_CURSOR	8	X'00000008'
MQSMPO_NONE	0	X'00000000'

MQSO_* (Subscribe Options):

MQSO_NONE	0	X'00000000'
MQSO_NON_DURABLE	0	X'00000000'
MQSO_READ_AHEAD_AS_Q_DEF	0	X'00000000'
MQSO_ALTER	1	X'00000001'
MQSO_CREATE	2	X'00000002'
MQSO_RESUME	4	X'00000004'
MQSO_DURABLE	8	X'00000008'
MQSO_GROUP_SUB	16	X'00000010'
MQSO_MANAGED	32	X'00000020'
MQSO_SET_IDENTITY_CONTEXT	64	X'00000040'

MQSO_FIXED_USERID	256	X'00000100'
MQSO_ANY_USERID	512	X'00000200'
MQSO_PUBLICATIONS_ON_REQUEST	2048	X'00000800'
MQSO_NEW_PUBLICATIONS_ONLY	4096	X'00001000'
MQSO_FAIL_IF_QUIESCING	8192	X'00002000'
MQSO_ALTERNATE_USER_AUTHORITY	262144	X'00040000'
MQSO_WILDCARD_CHAR	1048576	X'00100000'
MQSO_WILDCARD_TOPIC	2097152	X'00200000'
MQSO_SET_CORREL_ID	4194304	X'00400000'
MQSO_SCOPE_QMGR	67108864	X'04000000'
MQSO_NO_READ_AHEAD	134217728	X'08000000'
MQSO_READ_AHEAD	268435456	X'10000000'

MQSP_* (Sync point Availability):

MQSP_AVAILABLE	1	X'00000001'
MQSP_NOT_AVAILABLE	0	X'00000000'

MQSQQM_* (Shared Queue Queue Manager Name):

MQSQQM_USE	0	X'00000000'
MQSQQM_IGNORE	1	X'00000001'

MQSR_* (Action):

MQSR_ACTION_PUBLICATION	1	X'00000001'

MQSRO_* (Subscription request options structure):

MQSRO_STRUC_ID	"SROb"	
MQSRO_STRUC_ID_ARRAY	'S','R','O','b'	
MQSRO_VERSION_1	1	X'00000001'
MQSRO_CURRENT_VERSION	1	X'00000001'
MQSRO_NONE	0	X'00000000'
MQSRO_FAIL_IF_QUIESCING	8192	X'00002000'

MQSS_* (Segment Status):

MQSS_NOT_A_SEGMENT	'b'	
MQSS_SEGMENT	'S'	
MQSS_LAST_SEGMENT	'L'	

MQSSL_* (SSL FIPS Requirements):

MQSSL_FIPS_NO	0	X'00000000'
MQSSL_FIPS_YES	1	X'00000001'

MQSTAT_* (Stat Options):

MQSTAT_TYPE_ASYNC_ERROR	0	X'00000000'
MQSTAT_TYPE_RECONNECTION	0	X'00000000'
MQSTAT_TYPE_RECONNECTION_ERROR	0	X'00000000'

MQSTS_* (Status reporting structure structure):

MQSTS_STRUC_ID	"STAT"	
MQSTS_STRUC_ID_ARRAY	'S','T','A','T'	
MQSTS_VERSION_1	1	X'00000001'
MQSTS_CURRENT_VERSION	1	X'00000001'

MQSUB_* (Durable subscriptions):
Durable subscriptions

MQSUB_DURABLE_AS_PARENT	0	X'00000000'
MQSUB_DURABLE_ALLOWED	1	X'00000001'
MQSUB_DURABLE_INHIBITED	2	X'00000002'

Durable Subscriptions

MQSUB_DURABLE_ALL	-1	X'FFFFFFFF'
MQSUB_DURABLE_YES	1	X'00000001'
MQSUB_DURABLE_NO	2	X'00000002'

MQSUBTYPE_* (Command format Subscription Types):

MQSUBTYPE_API	1	X'00000001'
MQSUBTYPE_ADMIN	2	X'00000002'
MQSUBTYPE_PROXY	3	X'00000003'
MQSUBTYPE_ALL	-1	X'FFFFFFFF'
MQSUBTYPE_USER	-2	X'FFFFFFFE'

MQSUS_* (Command format Suspend Status):

MQSUS_YES	1	X'00000001'
MQSUS_NO	0	X'00000000'

MQSVC_* (Service):

Service Types

MQSVC_TYPE_COMMAND	0	X'00000000'
MQSVC_TYPE_SERVER	1	X'00000001'

Service Controls

MQSVC_CONTROL_Q_MGR	0	X'00000000'
MQSVC_CONTROL_Q_MGR_START	1	X'00000001'
MQSVC_CONTROL_MANUAL	2	X'00000002'

Service Status

MQSVC_STATUS_STOPPED	0	X'00000000'
MQSVC_STATUS_STARTING	1	X'00000001'
MQSVC_STATUS_RUNNING	2	X'00000002'
MQSVC_STATUS_STOPPING	3	X'00000003'
MQSVC_STATUS_RETRYING	4	X'00000004'

MQSYNCPOINT_* (Command format Syncpoint values for Pub/Sub migration):

MQSYNCPOINT_YES	0	X'00000000'
MQSYNCPOINT_IFPER	1	X'00000001'

MQSYSP_* (Command format System Parameter Values):

MQSYSP_NO	0	X'00000000'
MQSYSP_YES	1	X'00000001'
MQSYSP_EXTENDED	2	X'00000002'
MQSYSP_TYPE_INITIAL	10	X'0000000A'
MQSYSP_TYPE_SET	11	X'0000000B'
MQSYSP_TYPE_LOG_COPY	12	X'0000000C'
MQSYSP_TYPE_LOG_STATUS	13	X'0000000D'
MQSYSP_TYPE_ARCHIVE_TAPE	14	X'0000000E'
MQSYSP_ALLOC_BLK	20	X'00000014'
MQSYSP_ALLOC_TRK	21	X'00000015'
MQSYSP_ALLOC_CYL	22	X'00000016'
MQSYSP_STATUS_BUSY	30	X'0000001E'
MQSYSP_STATUS_PREMOUNT	31	X'0000001F'
MQSYSP_STATUS_AVAILABLE	32	X'00000020'
MQSYSP_STATUS_UNKNOWN	33	X'00000021'

MQSYSP_STATUS_ALLOC_ARCHIVE	34	X'00000022'
MQSYSP_STATUS_COPYING_BSDS	35	X'00000023'
MQSYSP_STATUS_COPYING_LOG	36	X'00000024'

MQTA_* (Topic attributes):
Wildcards

MQTA_BLOCK	1	X'00000001'
MQTA_PASSTHRU	2	X'00000002'

Subscriptions Allowed

MQTA_SUB_AS_PARENT	0	X'00000000'
MQTA_SUB_INHIBITED	1	X'00000001'
MQTA_SUB_ALLOWED	2	X'00000002'

Proxy Sub Propagation

MQTA_PROXY_SUB_FORCE	1	X'00000001'
MQTA_PROXY_SUB_FIRSTUSE	2	X'00000002'

Publications Allowed

MQTA_PUB_AS_PARENT	0	X'00000000'
MQTA_PUB_INHIBITED	1	X'00000001'
MQTA_PUB_ALLOWED	2	X'00000002'

MQTC_* (Trigger Controls):

MQTC_OFF	0	X'00000000'
MQTC_ON	1	X'00000001'

MQTCPKEEP_* (TCP Keepalive):

MQTCPKEEP_NO	0	X'00000000'
MQTCPKEEP_YES	1	X'00000001'

MQTCPSTACK_* (TCP Stack Types):

MQTCPSTACK_SINGLE	0	X'00000000'
MQTCPSTACK_MULTIPLE	1	X'00000001'

MQTIME_* (Command format Time units):

MQTIME_UNIT_MINS	0	X'00000000'
MQTIME_UNIT_SECS	1	X'00000001'

MQTM_* (Trigger message structure):

MQTM_STRUC_ID	"TMbb"	
MQTM_STRUC_ID_ARRAY	'T','M','b','b'	
MQTM_VERSION_1	1	X'00000001'
MQTM_CURRENT_VERSION	1	X'00000001'

MQTMC_* (Trigger message character format structure):

MQTMC_STRUC_ID	"TMCb"	
MQTMC_STRUC_ID_ARRAY	'T','M','C','b'	
MQTMC_VERSION_1	"bbb1"	
MQTMC_VERSION_2	"bbb2"	
MQTMC_CURRENT_VERSION	"bbb2"	
MQTMC_VERSION_1_ARRAY	'b','b','b','1'	
MQTMC_VERSION_2_ARRAY	'b','b','b','2'	
MQTMC_CURRENT_VERSION_ARRAY	'b','b','b','2'	

MQTOPT_* (Topic Type):

MQTOPT_LOCAL	0	X'00000000'
MQTOPT_CLUSTER	1	X'00000001'
MQTOPT_ALL	2	X'00000002'

MQTRAXSTR_* (Channel Initiator Trace Autostart):

MQTRAXSTR_NO	0	X'00000000'
MQTRAXSTR_YES	1	X'00000001'

MQTSCOPE_* (Subscription Scope):

MQTSCOPE_QMGR	1	X'00000001'
MQTSCOPE_ALL	2	X'00000002'

MQTT_* (Trigger Types):

MQTT_NONE	0	X'00000000'
MQTT_FIRST	1	X'00000001'
MQTT_EVERY	2	X'00000002'
MQTT_DEPTH	3	X'00000003'

MQTYPE_* (Property data types):

MQTYPE_AS_SET	0	X'00000000'
MQTYPE_NULL	2	X'00000002'
MQTYPE_BOOLEAN	4	X'00000004'
MQTYPE_BYTE_STRING	8	X'00000008'
MQTYPE_INT8	16	X'00000010'
MQTYPE_INT16	32	X'00000020'
MQTYPE_INT32	64	X'00000040'
MQTYPE_LONG	64	X'00000040'
MQTYPE_INT64	128	X'00000080'
MQTYPE_FLOAT32	256	X'00000100'
MQTYPE_FLOAT64	512	X'00000200'
MQTYPE_STRING	1024	X'00000400'

MQUA_* (Publish/Subscribe User Attribute Selectors):

MQUA_FIRST	65536	X'00010000'
MQUA_LAST	999999999	X'3B9AC9FF'

MQUIDSUPP_* (Command format User ID Support):

MQUIDSUPP_NO	0	X'00000000'
MQUIDSUPP_YES	1	X'00000001'

MQUNDELIVERED_* (Command format Undelivered values for Pub/Sub migration):

MQUNDELIVERED_NORMAL	0	X'00000000'
MQUNDELIVERED_SAFE	1	X'00000001'
MQUNDELIVERED_DISCARD	2	X'00000002'
MQUNDELIVERED_KEEP	3	X'00000003'

MQUOWST_* (Command format UOW States):

MQUOWST_NONE	0	X'00000000'
MQUOWST_ACTIVE	1	X'00000001'
MQUOWST_PREPARED	2	X'00000002'
MQUOWST_UNRESOLVED	3	X'00000003'

MQUOWT_* (Command format UOW Types):

MQUOWT_Q_MGR	0	X'00000000'
MQUOWT_CICS	1	X'00000001'
MQUOWT_RRS	2	X'00000002'
MQUOWT_IMS	3	X'00000003'
MQUOWT_XA	4	X'00000004'

MQUS_* (Queue Usages):

MQUS_NORMAL	0	X'00000000'
MQUS_TRANSMISSION	1	X'00000001'

MQUSAGE_* (Command format Page Set Usage Values and Data Set Usage Values):
Command format Page Set Usage Values

MQUSAGE_PS_AVAILABLE	0	X'00000000'
MQUSAGE_PS_DEFINED	1	X'00000001'
MQUSAGE_PS_OFFLINE	2	X'00000002'
MQUSAGE_PS_NOT_DEFINED	3	X'00000003'
MQUSAGE_EXPAND_USER	1	X'00000001'
MQUSAGE_EXPAND_SYSTEM	2	X'00000002'
MQUSAGE_EXPAND_NONE	3	X'00000003'

Command format Data Set Usage Values

MQUSAGE_DS_OLDEST_ACTIVE_UOW	10	X'0000000A'
MQUSAGE_DS_OLDEST_PS_RECOVERY	11	X'0000000B'
MQUSAGE_DS_OLDEST_CF_RECOVERY	12	X'0000000C'

MQVL_* (Value Length):

MQVL_NULL_TERMINATED	-1	X'FFFFFFFF'
MQVL_EMPTY_STRING	0	X'00000000'

MQVU_* (Variable User ID):

MQVU_FIXED_USER	1	X'00000001'
MQVU_ANY_USER	2	X'00000002'

MQWDR_* (Cluster workload exit destination record structure):

MQWDR_STRUC_ID	"WDRb"	
MQWDR_STRUC_ID_ARRAY	'W','D','R','b'	
MQWDR_VERSION_1	1	X'00000001'
MQWDR_VERSION_2	2	X'00000002'
MQWDR_CURRENT_VERSION	2	X'00000002'
MQWDR_LENGTH_1	124	X'0000007C'
MQWDR_LENGTH_2	136	X'00000088'
MQWDR_CURRENT_LENGTH	136	X'00000088'

MQWI_* (Wait Interval):

MQWI_UNLIMITED	-1	X'FFFFFFFF'

MQWIH_* (Workload information header structure and Flags):
Workload information header structure

MQWIH_STRUC_ID	"WIHb"	
MQWIH_STRUC_ID_ARRAY	'W','I','H','b'	
MQWIH_VERSION_1	1	X'00000001'
MQWIH_CURRENT_VERSION	1	X'00000001'
MQWIH_LENGTH_1	120	X'00000078'
MQWIH_CURRENT_LENGTH	120	X'00000078'

Workload information header Flags

MQWIH_NONE	0	X'00000000'

MQWQR_* (Cluster workload exit queue record structure):

MQWQR_STRUC_ID	"WQRb"	
MQWQR_STRUC_ID_ARRAY	'W','Q','R','b'	
MQWQR_VERSION_1	1	X'00000001'
MQWQR_VERSION_2	2	X'00000002'
MQWQR_VERSION_3	3	X'00000003'
MQWQR_CURRENT_VERSION	3	X'00000003'
MQWQR_LENGTH_1	200	X'000000C8'
MQWQR_LENGTH_2	208	X'000000D0'
MQWQR_LENGTH_3	212	X'000000D4'
MQWQR_CURRENT_LENGTH	212	X'000000D4'

MQWS_* (Wildcard Schema):

MQWS_DEFAULT	0	X'00000000'
MQWS_CHAR	1	X'00000001'
MQWS_TOPIC	2	X'00000002'

MQWXP_* (Cluster workload exit parameter structure):
MQWXP_* (Cluster workload exit parameter structure)

MQWXP_STRUC_ID	"WXPb"	
MQWXP_STRUC_ID_ARRAY	'W','X','P','b'	
MQWXP_VERSION_1	1	X'00000001'
MQWXP_VERSION_2	2	X'00000002'
MQWXP_VERSION_3	3	X'00000003'
MQWXP_VERSION_4	4	X'00000004'
MQWXP_CURRENT_VERSION	4	X'00000004'

MQWXP_* (Cluster Workload Flags)

MQWXP_PUT_BY_CLUSTER_CHL	2	X'00000002'

Related information:
Fields in MQWXP - Cluster workload exit parameter structure
Description of the fields in the MQWXP - Cluster workload exit parameter structure

MQXACT_* (API Caller Types):

MQXACT_EXTERNAL	1	X'00000001'
MQXACT_INTERNAL	2	X'00000002'

MQXC_* (Exit Commands):

MQXC_MQOPEN	1	X'00000001'
MQXC_MQCLOSE	2	X'00000002'
MQXC_MQGET	3	X'00000003'
MQXC_MQPUT	4	X'00000004'
MQXC_MQPUT1	5	X'00000005'
MQXC_MQINQ	6	X'00000006'
MQXC_MQSET	8	X'00000008'
MQXC_MQBACK	9	X'00000009'
MQXC_MQCMIT	10	X'0000000A'

MQXCC_* (Exit Responses):

MQXCC_OK	0	X'00000000'
MQXCC_SUPPRESS_FUNCTION	-1	X'FFFFFFFF'
MQXCC_SKIP_FUNCTION	-2	X'FFFFFFFE'
MQXCC_SEND_AND_REQUEST_SEC_MSG	-3	X'FFFFFFFD'
MQXCC_SEND_SEC_MSG	-4	X'FFFFFFFC'
MQXCC_SUPPRESS_EXIT	-5	X'FFFFFFFB'
MQXCC_CLOSE_CHANNEL	-6	X'FFFFFFFA'
MQXCC_REQUEST_ACK	-7	X'FFFFFFF9'
MQXCC_FAILED	-8	X'FFFFFFF8'

MQXDR_* (Exit Response):

MQXDR_OK	0	X'00000000'
MQXDR_CONVERSION_FAILED	1	X'00000001'

MQXE_* (Environments):

MQXE_OTHER	0	X'00000000'
MQXE_MCA	1	X'00000001'
MQXE_MCA_SVRCONN	2	X'00000002'
MQXE_COMMAND_SERVER	3	X'00000003'
MQXE_MQSC	4	X'00000004'

MQXEPO_* (Register Entry Point Options structure and Exit Options):
Register Entry Point Options structure

MQXEPO_STRUC_ID	"XEPO"	
MQXEPO_STRUC_ID_ARRAY	'X','E','P','O'	
MQXEPO_VERSION_1	1	X'00000001'
MQXEPO_CURRENT_VERSION	1	X'00000001'

Exit Options

MQXEPO_NONE	0	X'00000000'

MQXF_* (API Function Identifiers):

MQXF_INIT	1	X'00000001'
MQXF_TERM	2	X'00000002'
MQXF_CONN	3	X'00000003'
MQXF_CONNX	4	X'00000004'
MQXF_DISC	5	X'00000005'
MQXF_OPEN	6	X'00000006'
MQXF_CLOSE	7	X'00000007'

MQXF_PUT1	8	X'00000008'
MQXF_PUT	9	X'00000009'
MQXF_GET	10	X'0000000A'
MQXF_DATA_CONV_ON_GET	11	X'0000000B'
MQXF_INQ	12	X'0000000C'
MQXF_SET	13	X'0000000D'
MQXF_BEGIN	14	X'0000000E'
MQXF_CMIT	15	X'0000000F'
MQXF_BACK	16	X'00000010'
MQXF_STAT	18	X'00000012'
MQXF_CB	19	X'00000013'
MQXF_CTL	20	X'00000014'
MQXF_CALLBACK	21	X'00000015'
MQXF_SUB	22	X'00000016'
MQXF_SUBRQ	23	X'00000017'
MQXF_XACLOSE	24	X'00000018'
MQXF_XACOMMIT	25	X'00000019'
MQXF_XACOMPLETE	26	X'0000001A'
MQXF_XAEND	27	X'0000001B'
MQXF_XAFORGET	28	X'0000001C'
MQXF_XAOPEN	29	X'0000001D'
MQXF_XAPREPARE	30	X'0000001E'
MQXF_XARECOVER	31	X'0000001F'
MQXF_XAROLLBACK	32	X'00000020'
MQXF_XASTART	33	X'00000021'
MQXF_AXREG	34	X'00000022'
MQXF_AXUNREG	35	X'00000023'

MQXP_* (API crossing exit parameter structure):

MQXP_STRUC_ID	"XPbb"	
MQXP_STRUC_ID_ARRAY	'X','P','b','b'	
MQXP_VERSION_1	1	X'00000001'

MQXPDA_* (Problem Determination Area):

MQXPDA_NONE	X'00...00'	(48 nulls)
MQXPDA_NONE_ARRAY	'\0','\0',...	(48 nulls)

MQXPT_* (Transport Types):

MQXPT_ALL	-1	X'FFFFFFFF'
MQXPT_LOCAL	0	X'00000000'
MQXPT_LU62	1	X'00000001'
MQXPT_TCP	2	X'00000002'
MQXPT_NETBIOS	3	X'00000003'
MQXPT_SPX	4	X'00000004'
MQXPT_DECNET	5	X'00000005'
MQXPT_UDP	6	X'00000006'

MQXQH_* (Transmission queue header structure):

MQXQH_STRUC_ID	"XQHb"	
MQXQH_STRUC_ID_ARRAY	'X','Q','H','b'	
MQXQH_VERSION_1	1	X'00000001'
MQXQH_CURRENT_VERSION	1	X'00000001'

MQXR_* (Exit Reasons):

MQXR_BEFORE	1	X'00000001'
MQXR_AFTER	2	X'00000002'
MQXR_CONNECTION	3	X'00000003'
MQXR_INIT	11	X'0000000B'
MQXR_TERM	12	X'0000000C'
MQXR_MSG	13	X'0000000D'
MQXR_XMIT	14	X'0000000E'
MQXR_SEC_MSG	15	X'0000000F'
MQXR_INIT_SEC	16	X'00000010'
MQXR_RETRY	17	X'00000011'
MQXR_AUTO_CLUSSDR	18	X'00000012'
MQXR_AUTO_RECEIVER	19	X'00000013'
MQXR_CLWL_OPEN	20	X'00000014'
MQXR_CLWL_PUT	21	X'00000015'
MQXR_CLWL_MOVE	22	X'00000016'
MQXR_CLWL_REPOS	23	X'00000017'
MQXR_CLWL_REPOS_MOVE	24	X'00000018'
MQXR_END_BATCH	25	X'00000019'
MQXR_ACK_RECEIVED	26	X'0000001A'
MQXR_AUTO_SVRCONN	27	X'0000001B'

MQXR_AUTO_CLUSRCVR	28	X'0000001C'
MQXR_SEC_PARMS	29	X'0000001D'

MQXR2_* (Exit Response 2):

MQXR2_PUT_WITH_DEF_ACTION	0	X'00000000'
MQXR2_PUT_WITH_DEF_USERID	1	X'00000001'
MQXR2_PUT_WITH_MSG_USERID	2	X'00000002'
MQXR2_USE_AGENT_BUFFER	0	X'00000000'
MQXR2_USE_EXIT_BUFFER	4	X'00000004'
MQXR2_DEFAULT_CONTINUATION	0	X'00000000'
MQXR2_CONTINUE_CHAIN	8	X'00000008'
MQXR2_SUPPRESS_CHAIN	16	X'00000010'
MQXR2_STATIC_CACHE	0	X'00000000'
MQXR2_DYNAMIC_CACHE	32	X'00000020'

MQXT_* (Exit Identifiers):

MQXT_API_CROSSING_EXIT	1	X'00000001'
MQXT_API_EXIT	2	X'00000002'
MQXT_CHANNEL_SEC_EXIT	11	X'0000000B'
MQXT_CHANNEL_MSG_EXIT	12	X'0000000C'
MQXT_CHANNEL_SEND_EXIT	13	X'0000000D'
MQXT_CHANNEL_RCV_EXIT	14	X'0000000E'
MQXT_CHANNEL_MSG_RETRY_EXIT	15	X'0000000F'
MQXT_CHANNEL_AUTO_DEF_EXIT	16	X'00000010'
MQXT_CLUSTER_WORKLOAD_EXIT	20	X'00000014'
MQXT_PUBSUB_ROUTING_EXIT	21	X'00000015'

MQXUA_* (Exit User Area Value):

MQXUA_NONE	X'00...00'	(16 nulls)
MQXUA_NONE_ARRAY	'\0','\0',...	(16 nulls)

MQXWD_* (Exit wait descriptor structure):

MQXWD_STRUC_ID	"XWDb"	
MQXWD_STRUC_ID_ARRAY	'X','W','D','b'	
MQXWD_VERSION_1	1	X'00000001'

MQZAC_* (Application context structure):

MQZAC_STRUC_ID	"ZACb"	
MQZAC_STRUC_ID_ARRAY	'Z','A','C','b'	
MQZAC_VERSION_1	1	X'00000001'
MQZAC_CURRENT_VERSION	1	X'00000001'

MQZAD_* (Authority data structure):

MQZAD_STRUC_ID	"ZADb"	
MQZAD_STRUC_ID_ARRAY	'Z','A','D','b'	
MQZAD_VERSION_1	1	X'00000001'
MQZAD_VERSION_2	2	X'00000002'
MQZAD_CURRENT_VERSION	2	X'00000002'

MQZAET_* (Installable Services Entity Types):

MQZAET_NONE	0	X'00000000'
MQZAET_PRINCIPAL	1	X'00000001'
MQZAET_GROUP	2	X'00000002'
MQZAET_UNKNOWN	3	X'00000003'

MQZAO_* (Installable Services Authorizations):

MQZAO_CONNECT	1	X'00000001'
MQZAO_BROWSE	2	X'00000002'
MQZAO_INPUT	4	X'00000004'
MQZAO_OUTPUT	8	X'00000008'
MQZAO_INQUIRE	16	X'00000010'
MQZAO_SET	32	X'00000020'
MQZAO_PASS_IDENTITY_CONTEXT	64	X'00000040'
MQZAO_PASS_ALL_CONTEXT	128	X'00000080'
MQZAO_SET_IDENTITY_CONTEXT	256	X'00000100'
MQZAO_SET_ALL_CONTEXT	512	X'00000200'
MQZAO_ALTERNATE_USER_AUTHORITY	1024	X'00000400'
MQZAO_PUBLISH	2048	X'00000800'
MQZAO_SUBSCRIBE	4096	X'00001000'
MQZAO_RESUME	8192	X'00002000'
MQZAO_ALL_MQI	16383	X'00003FFF'
MQZAO_CREATE	65536	X'00010000'
MQZAO_DELETE	131072	X'00020000'
MQZAO_DISPLAY	262144	X'00040000'
MQZAO_CHANGE	524288	X'00080000'
MQZAO_CLEAR	1048576	X'00100000'
MQZAO_CONTROL	2097152	X'00200000'

MQZAO_CONTROL_EXTENDED	4194304	X'00400000'
MQZAO_AUTHORIZE	8388608	X'00800000'
MQZAO_ALL_ADMIN	16646144	X'00FE0000'
MQZAO_ALL	16662527	X'00FE3FFF'
MQZAO_REMOVE	16777216	X'01000000'
MQZAO_NONE	0	X'00000000'

MQZAS_* (Installable Services Service Interface Version):

MQZAS_VERSION_1	1	X'00000001'
MQZAS_VERSION_2	2	X'00000002'
MQZAS_VERSION_3	3	X'00000003'
MQZAS_VERSION_4	4	X'00000004'
MQZAS_VERSION_5	5	X'00000005'
MQZAS_VERSION_6	6	X'00000006'

MQZAT_* (Authentication Types):

MQZAT_INITIAL_CONTEXT	0	X'00000000'
MQZAT_CHANGE_CONTEXT	1	X'00000001'

MQZCI_* (Installable Services Continuation Indicator):

MQZCI_DEFAULT	0	X'00000000'
MQZCI_CONTINUE	0	X'00000000'
MQZCI_STOP	1	X'00000001'

MQZED_* (Entity data structure):

MQZED_STRUC_ID	"ZEDb"	
MQZED_STRUC_ID_ARRAY	'Z','E','D','b'	
MQZED_VERSION_1	1	X'00000001'
MQZED_VERSION_2	2	X'00000002'
MQZED_CURRENT_VERSION	2	X'00000002'

MQZFP_* (Free parameters structure):

MQZFP_STRUC_ID	"ZFPb"	
MQZFP_STRUC_ID_ARRAY	'Z','F','P','b'	
MQZFP_VERSION_1	1	X'00000001'
MQZFP_CURRENT_VERSION	1	X'00000001'

MQZIC_* (Identity context structure):

MQZIC_STRUC_ID	"ZICb"	
MQZIC_STRUC_ID_ARRAY	'Z','I','C','b'	
MQZIC_VERSION_1	1	X'00000001'
MQZIC_CURRENT_VERSION	1	X'00000001'

MQZID_* (Function ids for services):
Function ids common to all services

MQZID_INIT	0	X'00000000'
MQZID_TERM	1	X'00000001'

Function ids for Authority service

MQZID_INIT_AUTHORITY	0	X'00000000'
MQZID_TERM_AUTHORITY	1	X'00000001'
MQZID_CHECK_AUTHORITY	2	X'00000002'
MQZID_COPY_ALL_AUTHORITY	3	X'00000003'
MQZID_DELETE_AUTHORITY	4	X'00000004'
MQZID_SET_AUTHORITY	5	X'00000005'
MQZID_GET_AUTHORITY	6	X'00000006'
MQZID_GET_EXPLICIT_AUTHORITY	7	X'00000007'
MQZID_REFRESH_CACHE	8	X'00000008'
MQZID_ENUMERATE_AUTHORITY_DATA	9	X'00000009'
MQZID_AUTHENTICATE_USER	10	X'0000000A'
MQZID_FREE_USER	11	X'0000000B'
MQZID_INQUIRE	12	X'0000000C'
MQZID_CHECK_PRIVILEGED	13	X'0000000D'

Function ids for Name service

MQZID_INIT_NAME	0	X'00000000'
MQZID_TERM_NAME	1	X'00000001'
MQZID_LOOKUP_NAME	2	X'00000002'
MQZID_INSERT_NAME	3	X'00000003'
MQZID_DELETE_NAME	4	X'00000004'

Function ids for Userid service

MQZID_INIT_USERID	0	X'00000000'
MQZID_TERM_USERID	1	X'00000001'
MQZID_FIND_USERID	2	X'00000002'

MQZIO_* (Installable Services Initialization Options):

MQZIO_PRIMARY	0	X'00000000'
MQZIO_SECONDARY	1	X'00000001'

MQZNS_* (Name Service Interface Version):

MQZNS_VERSION_1	1	X'00000001'

MQZSE_* (Installable Services Start-Enumeration Indicator):

MQZSE_START	1	X'00000001'
MQZSE_CONTINUE	0	X'00000000'

MQZSL_* (Installable Services Selector Indicator):

MQZSL_NOT_RETURNED	0	X'00000000'
MQZSL_RETURNED	1	X'00000001'

MQZTO_* (Installable Services Termination Options):

MQZTO_PRIMARY	0	X'00000000'
MQZTO_SECONDARY	1	X'00000001'

MQZUS_* (Userid Service Interface Version):

MQZUS_VERSION_1	1	X'00000001'

Data types used in the MQI

Information on the data types that can be used in MQI. Descriptions, fields, and language declarations for relevant languages with each data type.

Introducing data types used in the MQI:

This section introduces the data types used in the MQI, and gives you some guidance on using them in the supported programming languages.

Elementary data types:

This section contains information about data types used in the MQI (or in exit functions). These are described in detail, followed by examples showing how to declare the elementary data types in the supported programming languages in the following topics.

The data types used in the MQI (or in exit functions) are either:

- Elementary data types, or
- Aggregates of elementary data types (arrays or structures)

The following elementary data types are used in the MQI (or in exit functions):

Elementary data type name	Data type	Description
MQBOOL	Boolean	The MQBOOL data type represents a boolean value. The value 0 represents false. Any other value represents true. An MQBOOL must be aligned as for the MQLONG data type.
MQBYTE	Byte	The MQBYTE data type represents a single byte of data. No particular interpretation is placed on the byte; it is treated as a string of bits, and not as a binary number or character. No special alignment is required. When MQBYTE data is sent between queue managers that use different character sets or encodings, the MQBYTE data is *not* converted in any way. The *MsgId* and *CorrelId* fields in the MQMD structure are like this. An array of MQBYTE is sometimes used to represent an area of main storage that is not known to the queue manager. For example, the area might contain application message data or a structure. The boundary alignment of this area must be compatible with the nature of the data contained within it. In the C programming language, any data type can be used for function parameters that are shown as arrays of MQBYTE. This is because such parameters are always passed by address, and in C the function parameter is declared as a pointer-to-void.

Elementary data type name	Data type	Description
MQBYTEn	String of *n* bytes	Each MQBYTEn data type represents a string of *n* bytes, where *n* can take any of the following values: 8, 16, 24, 32, 40, or 128. Each byte is described by the MQBYTE data type. No special alignment is required. If the data in the byte string is shorter than the defined length of the string, the data must be padded with nulls to fill the string. When the queue manager returns byte strings to the application (for example, on the MQGET call), the queue manager pads with nulls to the defined length of the string. Named constants are available to define the lengths of byte string fields. These are listed in “Constants” on page 1388
MQCHAR	Character	The MQCHAR data type represents a single-byte character, or one byte of a double-byte or multi-byte character. No special alignment is required. When MQCHAR data is sent between queue managers that use different character sets or encodings, the MQCHAR data usually requires conversion in order for the data to be interpreted correctly. The queue manager does this automatically for MQCHAR data in the MQMD structure. Conversion of MQCHAR data in the application message data is controlled by the MQGMO_CONVERT option specified on the MQGET call; see the description of this option in “MQGMO - Get-message options” on page 1655 for further details.

Elementary data type name	Data type	Description
MQCHARn	String of *n* characters	Each MQCHARn data type represents a string of *n* characters, where *n* can take any of the following values: 4, 8, 12, 20, 28, 32, 48, 64, 128, or 256. Each character is described by the MQCHAR data type. No special alignment is required. If the data in the string is shorter than the defined length of the string, the data must be padded with blanks to fill the string. In some cases a null character can be used to end the string prematurely, instead of padding with blanks; the null character and characters following it are treated as blanks, up to the defined length of the string. The places where a null can be used are identified in the call and data type descriptions. When the queue manager returns character strings to the application (for example, on the MQGET call), the queue manager always pads with blanks to the defined length of the string; the queue manager does not use the null character to delimit the string. Named constants are available that define the lengths of character string fields and are listed in "Constants" on page 1388.
MQFLOAT32	32-bit floating point number	The MQFLOAT32 data type is a 32-bit floating-point number represented using the standard IEEE floating-point format. An MQFLOAT32 must be aligned on a 4-byte boundary. The use of MQFLOAT32 in C on z/OS requires the use of the FLOAT(IEEE) compiler flag. The use of MQFLOAT32 in COBOL is limited to compilers that support floating-point numbers in IEEE format. This might require the use of the FLOAT(NATIVE) compiler flag.

Elementary data type name	Data type	Description
MQFLOAT64	64-bit floating point number	The MQFLOAT64 data type is a 64-bit floating-point number represented using the standard IEEE floating-point format. An MQFLOAT64 must be aligned on an 8-byte boundary. The use of MQFLOAT64 in C on z/OS requires the use of the FLOAT(IEEE) compiler flag. The use of MQFLOAT64 in COBOL is limited to compilers that support floating-point numbers in IEEE format. This might require the use of the FLOAT(NATIVE) compiler flag.
MQHCONFIG	Configuration handle	The MQHCONFIG data type represents a configuration handle, that is, the component that is being configured for a particular installable service. A configuration handle must be aligned on its natural boundary. Applications must not rely on the format of the data stored inside this handle. If valid, its value is intended to be usable in further MQI calls, but is not intended to have any meaning besides that purpose.
MQHCONN	Connection handle	The MQHCONN data type represents a connection handle, that is, the connection to a particular queue manager. A connection handle must be aligned on a 4-byte boundary. Applications must not rely on the format of the data stored inside this handle. If valid, its value is intended to be usable in further MQI calls, but is not intended to have any meaning besides that purpose.
MQHMSG	Message handle	The MQHMSG data type represents a message handle that gives access to a message. A message handle must be aligned on an 8-byte boundary. Applications must not rely on the format of the data stored inside this handle. If valid, its value is intended to be usable in further MQI calls, but is not intended to have any meaning besides that purpose.

Elementary data type name	Data type	Description
MQHOBJ	Object handle	The MQHOBJ data type represents an object handle that gives access to an object. An object handle must be aligned on a 4-byte boundary. Applications must not rely on the format of the data stored inside this handle. If valid, its value is intended to be usable in further MQI calls, but is not intended to have any meaning besides that purpose.
MQINT8	8-bit signed integer	The MQINT8 data type is an 8-bit signed integer that can take any value in the range -128 to +127, unless otherwise restricted by the context.
MQINT16	16-bit signed integer	The MQINT16 data type is a 16-bit signed integer that can take any value in the range -32 768 to +32 767, unless otherwise restricted by the context. An MQINT16 must be aligned on a 2-byte boundary.
MQINT32	32-bit signed integer	The MQINT32 data type is a 32-bit signed binary integer that can take any value in the range -2 147 483 648 through +2 147 483 647, unless otherwise restricted by the context. See the definition of MQLONG.
MQINT64	64-bit signed integer	The MQINT64 data type is a 64-bit signed integer that can take any value in the range -9 223 372 036 854 775 808 through +9 223 372 036 854 775 807, unless otherwise restricted by the context. For COBOL, the valid range is limited to -999 999 999 999 999 999 through +999 999 999 999 999 999. A 64-bit integer must be aligned on an 8-byte boundary.
MQLONG	32-bit signed integer	The MQLONG data type is a 32-bit signed binary integer that can take any value in the range -2 147 483 648 through +2 147 483 647, unless otherwise restricted by the context. For COBOL, the valid range is limited to -999 999 999 through +999 999 999. An MQLONG must be aligned on a 4-byte boundary.

Elementary data type name	Data type	Description
MQPID	Process identifier	The WebSphere MQ process identifier. This is the same identifier used in MQ trace and FFST™ dumps, but might be different from the operating system process identifier.
MQPTR	Pointer	The MQPTR data type is the address of data of any type. A pointer must be aligned on its natural boundary; this is a 16-byte boundary on IBM i, and an 8-byte boundary on other platforms. Some programming languages support typed pointers; the MQI also uses these in a few cases (for example, PMQCHAR and PMQLONG in the C programming language).
MQTID	Thread identifier	The WebSphere MQ thread identifier. This is the same identifier used in MQ trace and FFST™ dumps, but might be different from the operating system thread identifier.
MQUINT8	8-bit unsigned integer	The MQUINT8 data type is an 8-bit unsigned integer that can take any value in the range 0 to +255, unless otherwise restricted by the context.
MQUINT16	16-bit unsigned integer	The MQUINT16 data type is a 16-bit unsigned integer that can take any value in the range 0 through +65 535, unless otherwise restricted by the context. An MQUINT16 must be aligned on a 2-byte boundary.
MQUINT32	32-bit unsigned integer	The MQUINT32 data type is a 32-bit unsigned binary integer. See the definition of MQULONG.
MQUINT64	64-bit unsigned integer	The MQINT64 data type is a 64-bit unsigned integer that can take any value in the range 0 through +18 446 744 073 709 551 615, unless otherwise restricted by the context. For COBOL, the valid range is limited to 0 through +999 999 999 999 999 999. A 64-bit integer must be aligned on an 8-byte boundary.

Elementary data type name	Data type	Description
MQULONG	32-bit unsigned integer	The MQULONG data type is a 32-bit unsigned binary integer that can take any value in the range 0 through +4 294 967 294, unless otherwise restricted by the context. For COBOL, the valid range is limited to 0 through +999 999 999. An MQULONG must be aligned on a 4-byte boundary.
PMQACH	Pointer	Pointer to a data structure of type MQACH
PMQAIR	Pointer	Pointer to a data structure of type MQAIR
PMQAXC	Pointer	Pointer to a data structure of type MQAXC
PMQAXP	Pointer	Pointer to a data structure of type MQAXP
PMQBMHO	Pointer	Pointer to a data structure of type MQBMHO
PMQBO	Pointer	Pointer to a data structure of type MQBO
PMQBOOL	Pointer	Pointer to data of type MQBOOL
PMQBYTE	Pointer	Pointer to data of type MQBYTE
PMQBYTEn	Pointer	Pointer to data of type MQBYTEn, where n can be 8, 16, 24, 32, 40, 128
PMQCBC	Pointer	Pointer to a data structure of type MQCBC
PMQCBD	Pointer	Pointer to a data structure of type MQCBD
PMQCHAR	Pointer	Pointer to data of type MQCHAR
PMQCHARN	Pointer	Pointer to a data type of MQCHARN, where n can be 4, 8, 12, 20, 28, 32, 48, 64, 128, 256, 264
PMQCHARV	Pointer	Pointer to a data structure of type MQCHARV
PMQCIH	Pointer	Pointer to a data structure of type MQCIH
PMQCMHO	Pointer	Pointer to a data structure of type MQCMHO
PMQCNO	Pointer	Pointer to a data structure of type MQCNO
PMQCSP	Pointer	Pointer to a data structure of type MQCSP
PMQCTLO	Pointer	Pointer to a data structure of type MQCTLO
PMQDH	Pointer	Pointer to a data structure of type MQDH

Elementary data type name	Data type	Description
PMQDHO	Pointer	Pointer to a data structure of type MQDHO
PMQDLH	Pointer	Pointer to a data structure of type MQDLH
PMQDMHO	Pointer	Pointer to a data structure of type MQDMHO
PMQDMPO	Pointer	Pointer to a data structure of type MQDMPO
PMQEPH	Pointer	Pointer to a data structure of type MQEPH
PMQFLOAT32	Pointer	Pointer to a data structure of type MQFLOAT32
PMQFLOAT64	Pointer	Pointer to a data structure of type MQFLOAT64
PMQFUNC	Pointer	Pointer to a function
PMQGMO	Pointer	Pointer to a data structure of type MQGMO
PMQHCONFIG	Pointer	Pointer to data of type MQHCONFIG
PMQHCONN	Pointer	Pointer to data of type MQHCONN
PMQHMSG	Pointer	Pointer to data of type MQHMSG
PMQHOBJ	Pointer	Pointer to data of type MQHOBJ
PMQIIH	Pointer	Pointer to a data structure of type MQIIH
PMQIMPO	Pointer	Pointer to a data structure of type MQIMPO
PMQINT8	Pointer	Pointer to data of type MQINT8
PMQINT16	Pointer	Pointer to data of type MQINT16
PMQINT32	Pointer	Pointer to data of type MQINT32
PMQINT64	Pointer	Pointer to data of type MQINT64
PMQLONG	Pointer	Pointer to data of type MQLONG
PMQMD	Pointer	Pointer to structure of type MQMD
PMQMDE	Pointer	Pointer to a data structure of type MQMDE
PMQMD1	Pointer	Pointer to a data structure of type MQMD1
PMQMD2	Pointer	Pointer to a data structure of type MQMD2
PMQMHBO	Pointer	Pointer to a data structure of type MQMHBO
PMQOD	Pointer	Pointer to a data structure of type MQOD
PMQOR	Pointer	Pointer to a data structure of type MQOR
PMQPD	Pointer	Pointer to a data structure of type MQPD
PMQPID	Pointer	Pointer to a process identifier

Elementary data type name	Data type	Description
PMQMD	Pointer	Pointer to a data structure of type MQMD
PMQPMO	Pointer	Pointer to a data structure of type MQPMO
PMQPTR	Pointer	Pointer to data of type MQPTR
PMQRFH	Pointer	Pointer to a data structure of type MQRFH
PMQRFH2	Pointer	Pointer to a data structure of type MQRFH2
PMQRMH	Pointer	Pointer to a data structure of type MQRMH
PMQRR	Pointer	Pointer to a data structure of type MQRR
PMQSCO	Pointer	Pointer to a data structure of type MQSCO
PMQSD	Pointer	Pointer to a data structure of type MQSD
PMQSMPO	Pointer	Pointer to a data structure of type MQSMPO
PMQSRO	Pointer	Pointer to a data structure of type MQSRO
PMSSTS	Pointer	Pointer to a data structure of type MQSTS
PMQTID	Pointer	Pointer to a thread ID
PMQTM	Pointer	Pointer to a data structure of type MQTM
PMQTMC2	Pointer	Pointer to a data structure of type MQTMC2
PMQUINT8	Pointer	Pointer to a data type of MQUINT8
PMQUINT16	Pointer	Pointer to a data type of MQUINT16
PMQUINT32	Pointer	Pointer to a data type of MQUINT32
PMQUINT64	Pointer	Pointer to a data type of MQUINT64
PMQULONG	Pointer	Pointer to a data type of MQULONG
PMQVOID	Pointer	
PMQWIH	Pointer	Pointer to a data structure of type MQWIH
PMQXQH	Pointer	Pointer to a data structure of type MQXQH

C declarations:

Data type	Representation
MQBOOL	`typedef MQLONG MQBOOL;`
MQBYTE	`typedef unsigned char MQBYTE;`
MQBYTE8	`typedef MQBYTE MQBYTE8[8];`
MQBYTE16	`typedef MQBYTE MQBYTE16[16];`
MQBYTE24	`typedef MQBYTE MQBYTE24[24];`
MQBYTE32	`typedef MQBYTE MQBYTE32[32];`
MQBYTE40	`typedef MQBYTE MQBYTE40[40];`
MQCHAR	`typedef char MQCHAR;`
MQCHAR4	`typedef MQCHAR MQCHAR4[4];`
MQCHAR8	`typedef MQCHAR MQCHAR8[8];`
MQCHAR12	`typedef MQCHAR MQCHAR12[12];`
MQCHAR20	`typedef MQCHAR MQCHAR20[20];`
MQCHAR28	`typedef MQCHAR MQCHAR28[28];`
MQCHAR32	`typedef MQCHAR MQCHAR32[32];`
MQCHAR48	`typedef MQCHAR MQCHAR48[48];`
MQCHAR64	`typedef MQCHAR MQCHAR64[64];`
MQCHAR128	`typedef MQCHAR MQCHAR128[128];`
MQCHAR256	`typedef MQCHAR MQCHAR256[256];`
MQFLOAT32	`typedef float MQFLOAT32;`
MQFLOAT64	`typedef double MQFLOAT64;`
MQHCONFIG	`typedef void MQPOINTER MQHCONFIG;`
MQHCONN	`typedef MQLONG MQHCONN;`
MQHOBJ	`typedef MQLONG MQHOBJ;`
MQINT8	`typedef signed char MQINT8;`
MQINT16	`typedef short MQINT16;`

MQINT64	On 64-bit UNIX systems:
	`typedef long;`
	
	On 32-bit AIX, Solaris, and HP-UX:
	`typedef int64_t;`
	
	On IBM i, Linux, and z/OS:
	`typedef long long;`
	
	On Windows:
	`typedef _int64;`

MQLONG	On IBM i:
	`typedef long MQLONG;`
	
	other platforms:
	`if defined(MQ_64_BIT)`
	` typedef int MQLONG;`
	`else`
	` typedef long MQLONG;`

Data type	Representation
MQPID	`typedef MQLONG MQPID;`
MQPTR	`typedef void MQPOINTER MQPTR;`
MQTID	`typedef MQLONG MQTID;`
MQUINT8	`typedef unsigned char MQUINT8;`
MQUINT16	`typedef unsigned short MQUINT16;`

MQUINT64	On 64-bit UNIX systems:
	`typedef unsigned long;`
	
	On 32-bit AIX, Solaris, and HP-UX:
	`typedef uint64_t;`
	
	On IBM i, Linux, and z/OS:
	`typedef unsigned long long;`
	
	On Windows:
	`typedef unsigned _int64;`

MQULONG	On IBM i:
	`typedef unsigned long MQULONG;`
	
	other platforms:
	`if defined(MQ_64_BIT)`
	`  typedef unsigned int MQULONG;`
	`else`
	`  typedef unsigned long MQULONG;`

PMQBO	`typedef MQBO MQPOINTER PMQBO;`
PMQBOOL	`typedef MQBOOL MQPOINTER PMQBOOL;`
PMQBYTE	`typedef MQBYTE MQPOINTER PMQBYTE;`
PMQBYTE8	`typedef MQBYTE8[8] MQPOINTER PMQBYTE8[8];`
PMQBYTE16	`typedef MQBYTE16[16] MQPOINTER PMQBYTE16[16];`
PMQBYTE24	`typedef MQBYTE24[24] MQPOINTER PMQBYTE24[24];`
PMQBYTE32	`typedef MQBYTE32[32] MQPOINTER PMQBYTE32[32];`
PMQBYTE40	`typedef MQBYTE40[40] MQPOINTER PMQBYTE40[40];`
PMQBYTE128	`typedef MQBYTE128[128] MQPOINTER PMQBYTE128[128];`
PMQCHAR	`typedef MQCHAR MQPOINTER PMQCHAR;`
PMQCHAR4	`typedef MQCHAR4[4] MQPOINTER PMQCHAR4[4];`
PMQCHAR8	`typedef MQCHAR8[8] MQPOINTER PMQCHAR8[8];`
PMQCHAR12	`typedef MQCHAR12[12] MQPOINTER PMQCHAR12[12];`
PMQCHAR20	`typedef MQCHAR20[20] MQPOINTER PMQCHAR20[20];`
PMQCHAR28	`typedef MQCHAR28[28] MQPOINTER PMQCHAR28[28];`
PMQCHAR32	`typedef MQCHAR32[32] MQPOINTER PMQCHAR32[32];`
PMQCHAR48	`typedef MQCHAR48[48] MQPOINTER PMQCHAR48[48];`
PMQCHAR64	`typedef MQCHAR64[64] MQPOINTER PMQCHAR64[64];`
PMQCHAR128	`typedef MQCHAR128[128] MQPOINTER PMQCHAR128[128];`
PMQCHAR256	`typedef MQCHAR256[256] MQPOINTER PMQCHAR256[256];`
PMQCHAR264	`typedef MQCHAR264[264] MQPOINTER PMQCHAR264[264];`

Data type	Representation
PMQCIH	`typedef MQCIH MQPOINTER PMQCIH;`
PMQCNO	`typedef MQCNO MQPOINTER PMQCNO;`
PMQDLH	`typedef MQDLH MQPOINTER PMQDLH;`
PMQFUNC	`typedef void MQPOINTER PMQFUNC;`
PMQFLOAT32	`typedef MQFLOAT32 MQPOINTER PMQFLOAT32;`
PMQFLOAT64	`typedef MQFLOAT64 MQPOINTER PMQFLOAT64;`
PMQGMO	`typedef MQGMO MQPOINTER PMQGMO;`
PMQHCONFIG	`typedef MQHCONFIG MQPOINTER PMQHCONFIG;`
PMQHCONN	`typedef MQHCONN MQPOINTER PMQHCONN;`
PMQHOBJ	`typedef MQHOBJ MQPOINTER PMQHOBJ;`
PMQIIH	`typedef MQIIH MQPOINTER PMQIIH;`
PMQINT8	`typedef MQINT8 MQPOINTER PMQINT8;`
PMQINT16	`typedef MQINT16 MQPOINTER PMQINT16;`
PMQLONG	`typedef MQLONG MQPOINTER PMQLONG;`
PMQMD	`typedef MQMD MQPOINTER PMQMD;`
PMQMD1	`typedef MQMD1[1] MQPOINTER PMQMD1[1];`
PMQMDE	`typedef MQMDE MQPOINTER PMQMDE;`
PMQOD	`typedef MQOD MQPOINTER PMQOD;`
PMQPMO	`typedef MQPMO MQPOINTER PMQPMO;`
PMQPTR	`typedef MQPTR MQPOINTER PMQPTR;`
PMQRFH	`typedef MQRFH MQPOINTER PMQRFH;`
PMQRFH2	`typedef MQRFH2[2] MQPOINTER PMQRFH2[2];`
PMQRMH	`typedef MQRMH MQPOINTER PMQRMH;`
PMQTM	`typedef MQTM MQPOINTER PMQTM;`
PMQTMC2	`typedef MQTMC2[2] MQPOINTER PMQTMC2[2];`
PMQUINT8	`typedef MQUINT8 MQPOINTER PMQUINT8;`
PMQUINT16	`typedef MQUINT16 MQPOINTER PMQUINT16;`
PMQULONG	`typedef MQULONG MQPOINTER PMQULONG;`
PMQVOID	`typedef void MQPOINTER PMQVOID;`
PMQWIH	`typedef MQWIH MQPOINTER PMQWIH;`
PMQXQH	`typedef MQXQH MQPOINTER PMQXQH;`
PPMQBO	`typedef PMQBO MQPOINTER PPMQBO;`
PPMQBYTE	`typedef PMQBYTE MQPOINTER PPMQBYTE;`
PPMQCHAR	`typedef PMQCHAR MQPOINTER PPMQCHAR;`
PPMQCNO	`typedef PMQCNO MQPOINTER PPMQCNO;`
PPMQGMO	`typedef PMQGMO MQPOINTER PPMQGMO;`
PPMQHCONN	`typedef PMQHCONN MQPOINTER PPMQHCONN;`
PPMQHOBJ	`typedef PMQHOBJ MQPOINTER PPMQHOBJ;`
PPMQLONG	`typedef PMQLONG MQPOINTER PPMQLONG;`
PPMQMD	`typedef PMQMD MQPOINTER PPMQMD;`
PPMQOD	`typedef PMQOD MQPOINTER PPMQOD;`

Data type	Representation
PPMQPMO	`typedef PMQPMO MQPOINTER PPMQPMO;`
PPMQULONG	`typedef PMQULONG MQPOINTER PPMQULONG;`
PPMQVOID	`typedef PMQVOID MQPOINTER PPMQVOID;`
Where `defined(MQ_64_BIT)` means a 64 bit platform.	

See "Data types" on page 1549 for a description of the MQPOINTER macro variable.

COBOL declarations:

Data type	Representation
MQBOOL	`PIC S9(9) BINARY`
MQBYTE	`PIC X`
MQBYTE8	`PIC X(8)`
MQBYTE16	`PIC X(16)`
MQBYTE24	`PIC X(24)`
MQBYTE32	`PIC X(32)`
MQBYTE40	`PIC X(40)`
MQCHAR	`PIC X`
MQCHAR4	`PIC X(4)`
MQCHAR8	`PIC X(8)`
MQCHAR12	`PIC X(12)`
MQCHAR20	`PIC X(20)`
MQCHAR28	`PIC X(28)`
MQCHAR32	`PIC X(32)`
MQCHAR48	`PIC X(48)`
MQCHAR64	`PIC X(64)`
MQCHAR128	`PIC X(128)`
MQCHAR256	`PIC X(256)`
MQFLOAT32	`USAGE COMP-1`
MQFLOAT64	`USAGE COMP-2`
MQHCONN	`PIC S9(9) BINARY`
MQHOBJ	`PIC S9(9) BINARY`
MQINT8	`PIC S9(2) BINARY`
MQINT16	`PIC S9(4) BINARY`
MQINT64	`PIC S9(18) BINARY`
MQLONG	`PIC S9(9) BINARY`
MQPTR	`POINTER`
MQUINT8	`PIC 9(2) BINARY`
MQUINT16	`PIC 9(4) BINARY`
MQUINT64	`PIC 9(18) BINARY`
MQULONG	`PIC 9(9) BINARY`

PL/I declarations:

PL/I is supported on z/OS.

Data type	Representation
MQBOOL	`fixed bin(31)`
MQBYTE	`char(1)`
MQBYTE8	`char(8)`
MQBYTE16	`char(16)`
MQBYTE24	`char(24)`
MQBYTE32	`char(32)`
MQBYTE40	`char(40)`
MQCHAR	`char(1)`
MQCHAR4	`char(4)`
MQCHAR8	`char(8)`
MQCHAR12	`char(12)`
MQCHAR20	`char(20)`
MQCHAR28	`char(28)`
MQCHAR32	`char(32)`
MQCHAR48	`char(48)`
MQCHAR64	`char(64)`
MQCHAR128	`char(128)`
MQCHAR256	`char(256)`
MQFLOAT32	`binary float(21) ieee`
MQFLOAT64	`binary float(52) ieee`
MQHCONN	`fixed bin(31)`
MQHOBJ	`fixed bin(31)`
MQINT8	`fixed bin(7)`
MQINT16	`fixed bin(15)`
MQINT64	`fixed bin(63)`
MQLONG	`fixed bin(31)`
MQPTR	`pointer`
MQUINT8	`fixed bin(8)`
MQUINT16	`fixed bin(16)`
MQUINT64	`fixed bin(64)`
MQULONG	`fixed bin(32)`

System/390 assembler declarations:

System/390 assembler is supported on z/OS only.

Data type	Representation
MQBOOL	DS F
MQBYTE	DS XL1
MQBYTE8	DS XL8
MQBYTE16	DS XL16
MQBYTE24	DS XL24
MQBYTE32	DS XL32
MQBYTE40	DS XL40
MQCHAR	DS CL1
MQCHAR4	DS CL4
MQCHAR8	DS CL8
MQCHAR12	DS CL12
MQCHAR20	DS CL20
MQCHAR28	DS CL28
MQCHAR32	DS CL32
MQCHAR48	DS CL48
MQCHAR64	DS CL64
MQCHAR128	DS CL128
MQCHAR256	DS CL256
MQFLOAT32	DS EB
MQFLOAT64	DS DB
MQHCONN	DS F
MQHOBJ	DS F
MQINT8	DS XL1
MQINT16	DS H
MQINT64	DS D
MQLONG	DS F
MQPTR	DS F
MQUINT8	DS XL1
MQUINT16	DS H
MQUINT64	DS D
MQULONG	DS F

Structure data types - introduction:

This section introduces the structure data types used in the MQI. The structure data types themselves are described in subsequent sections.

Summary:

The following tables summarize the structure data types used in the MQI.

Table 115. Structure data types used on MQI calls (or exit functions):

Structure	Description	Calls where used
MQACH	API exit chain header	
MQAIR	Authentication information record	MQCONNX
MQAXC	API exit context	
MQAXP	API exit parameter	
MQBMHO	Buffer to message handle options	MQBUFMH
MQBO	Begin options	MQBEGIN
MQCBD	Callback descriptor	MQCB
MQCBO	Create-bag options	mqCreateBag
MQCHARV	Variable length string	MQINQMP
MQCNO	Connect options	MQCONNX
MQCSP	Security parameters	MQCONNX
MQCTLO	Callback options	MQCTL
MQDMPO	Delete message property options	MQDLTMP
MQGMO	Get-message options	MQGET
MQIMPO	Inquire message property options	MQINQMP
MQMD	Message descriptor	MQBUFMH, MQMHBUF, MQCB, MQGET, MQPUT, MQPUT1
MQMHBO	Message handle to buffer options	MQMHBUF
MQOD	Object descriptor	MQOPEN, MQPUT1
MQOR	Object record	MQOPEN, MQPUT1
MQPD	Property descriptor	MQSETMP
MQPMO	Put-message options	MQPUT, MQPUT1
MQPMR	Put-message record	MQPUT, MQPUT1
MQRR	Response record	MQOPEN, MQPUT, MQPUT1
MQSCO	SSL configuration options	MQCONNX
MQSD	Subscription descriptor	MQSUB
MQSMPO	Set message property option	MQSETMP
MQSRO	Subscription request options	MQSUBRQ
MQSTS	Status reporting structure	MQSTAT

Table 116. Structure data types used in message data:

Structure	Description
MQCIH	CICS information header
MQCFH	PCF header
MQEPH	Embedded PCF header
MQDH	Distribution header
MQDLH	Dead letter (undelivered message) header
MQIIH	IMS information header
MQMDE	Message descriptor extension
MQRFH	Rules and formatting header
MQRFH2	Rules and formatting header 2
MQRMH	Reference message header
MQTM	Trigger message
MQTMC2	Trigger message (character format 2)
MQWIH	Work information header
MQXQH	Transmission queue header

Note: The MQDXP structure (data conversion exit parameter) is described in "Data conversion" on page 2210, together with the associated data conversion calls.

Rules for structure data types:

Programming languages vary in their level of support for structures, and certain rules and conventions are adopted to map the MQI structures consistently in each programming language:

1. Structures must be aligned on their natural boundaries.
   - Most MQI structures require 4-byte alignment.
   - On IBM i, structures containing pointers require 16-byte alignment; these are: MQCNO, MQOD, MQPMO.
2. Each field in a structure must be aligned on its natural boundary.
   - Fields with data types that equate to MQLONG must be aligned on 4-byte boundaries.
   - Fields with data types that equate to MQPTR must be aligned on 16-byte boundaries on IBM i, and 4-byte boundaries in other environments.
   - Other fields are aligned on 1-byte boundaries.
3. The length of a structure must be a multiple of its boundary alignment.
   - Most MQI structures have lengths that are multiples of 4 bytes.
   - On IBM i, structures containing pointers have lengths that are multiples of 16 bytes.
4. Where necessary, padding bytes or fields must be added to ensure compliance with the above rules.

Conventions used in the descriptions:

The description of each structure data type includes:

- An overview of the purpose and use of the structure
- Descriptions of the fields in the structure, in a form that is independent of the programming language
- Examples of how the structure is declared in each of the supported programming languages

The description of each structure data type contains the following sections:

Structure name
The name of the structure, followed by a summary of the fields in the structure.

Overview
A brief description of the purpose and use of the structure.

Fields Descriptions of the fields. For each field, the name of the field is followed by its elementary data type in parentheses ( ). In text, field names are shown using an italic typeface; for example *Version*.

There is also a description of the purpose of the field, together with a list of any values that the field can take. Names of constants are shown in uppercase; for example MQGMO_STRUC_ID. A set of constants having the same prefix is shown using the * character, for example: MQIA_*.

In the descriptions of the fields, the following terms are used:

input You supply information in the field when you make a call.

output
The queue manager returns information in the field when the call completes or fails.

input/output
You supply information in the field when you make a call, and the queue manager changes the information when the call completes or fails.

Initial values
A table showing the initial values for each field in the data definition files supplied with the MQI.

C declaration
Typical declaration of the structure in C.

COBOL declaration
Typical declaration of the structure in COBOL.

PL/I declaration
Typical declaration of the structure in PL/I.

System/390 assembler declaration
Typical declaration of the structure in System/390 assembler language.

Visual Basic declaration
Typical declaration of the structure in Visual Basic.

C programming:

This section contains information to help you use the MQI from the C programming language.

Header files:

Header files are provided to help you write C application programs that use the MQI.

These header files are summarized in Table 117.

Table 117. C header files

File	Contents
CMQC	Function prototypes, data types, and named constants for the main MQI
CMQXC	Function prototypes, data types, and named constants for the data conversion exit
CMQEC	Function prototypes, data types, and named constants for the main MQI, data conversion exit and Interface Entry Points structure (CMQEC includes CMQXC and CMQC.)

To improve the portability of applications, code the name of the header file in lowercase on the **#include** preprocessor directive:

```
#include "cmqec.h"
```

Functions:

You do not need to specify all parameters that are passed by address every time you invoke a function.

- Pass parameters that are *input-only* and of type MQHCONN, MQHOBJ, or MQLONG by value.
- Pass all other parameters by address.

Where a particular parameter is not required, use a null pointer as the parameter on the function invocation, in place of the address of the parameter data. Parameters for which this is possible are identified in the call descriptions.

No parameter is returned as the value of the function; in C terminology, this means that all functions return **void**.

The attributes of the function are defined by the MQENTRY macro variable; the value of this macro variable depends on the environment.

Parameters with undefined data type:

The *Buffer* parameter on the MQGET, MQPUT, and MQPUT1 functions has an undefined data type. This parameter is used to send and receive the application's message data.

Parameters of this sort are shown in the C examples as arrays of MQBYTE. You can declare the parameters in this way, but it is usually more convenient to declare them as the particular structure that describes the layout of the data in the message. Declare the actual function parameter as a pointer-to-void, and specify the address of any sort of data as the parameter on the function invocation.

Data types:

Define all data types using the C **typedef** statement. For each data type, also define the corresponding pointer data type. The name of the pointer data type is the name of the elementary or structure data type prefixed with the letter P to denote a pointer. Define the attributes of the pointer using the MQPOINTER macro variable; the value of this macro variable depends on the environment. The following illustrates how to declare pointer data types:

```
#define MQPOINTER *               /* depends on environment */
...
typedef MQLONG MQPOINTER PMQLONG;  /* pointer to MQLONG */
typedef MQMD   MQPOINTER PMQMD;    /* pointer to MQMD */
```

Manipulating binary strings:

Declare strings of binary data as one of the MQBYTEn data types.

Whenever you copy, compare, or set fields of this type, use the C functions **memcpy**, **memcmp**, or **memset**; for example:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set "MsgId" field to nulls    */
       MQMI_NONE,                 /* ...using named constant      */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set "CorrelId" field to nulls */
       0x00,                      /* ...using a different method   */
       sizeof(MQBYTE24));
```

Do not use the string functions **strcpy**, **strcmp**, **strncpy**, or **strncmp**, because these do not work correctly for data declared with the MQBYTEn data types.

Manipulating character strings:

When the queue manager returns character data to the application, the queue manager always pads the character data with blanks to the defined length of the field; the queue manager *does not* return null-terminated strings.

Therefore, when copying, comparing, or concatenating such strings, use the string functions **strncpy**, **strncmp**, or **strncat**.

Do not use the string functions that require the string to be terminated by a null (**strcpy**, **strcmp**, **strcat**). Also, do not use the function **strlen** to determine the length of the string; use instead the **sizeof** function to determine the length of the field.

Initial values for structures:

The header files define various macro variables that you can use to provide initial values for the MQ structures when you declare instances of those structures.

These macro variables have names of the form MQxxx_DEFAULT, where MQxxx represents the name of the structure. They are used in the following way:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

For some character fields (for example, the *StrucId* fields that occur in most structures, or the *Format* field that occurs in MQMD), the MQI defines particular values that are valid. For each of the valid values, *two* macro variables are provided:

- One macro variable defines the value as a string with a length, excluding the implied null matches, exactly the defined length of the field. For example, for the *Format* field in MQMD the following macro variable is provided (¬ represents a blank character):

  ```
  #define MQFMT_STRING "MQSTR¬¬¬"
  ```

  Use this form with the **memcpy** and **memcmp** functions.
- The other macro variable defines the value as an array of characters; the name of this macro variable is the name of the string form suffixed with _ARRAY. For example:

  ```
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','⌂','⌂','⌂'
  ```

  Use this form to initialize the field when you declare an instance of the structure with values different from those provided by the MQMD_DEFAULT macro variable. (This is not always necessary; in some environments you can use the string form of the value in both situations. However, you can use the array form for declarations, because this is required for compatibility with the C++ programming language.)

Initial values for dynamic structures:

When a variable number of instances of a structure is required, the instances are typically created in main storage obtained dynamically using the **calloc** or **malloc** functions. To initialize the fields in such structures, consider the following technique:

1. Declare an instance of the structure using the appropriate MQxxx_DEFAULT macro variable to initialize the structure. This instance becomes the model for other instances:

   ```
   MQMD Model = {MQMD_DEFAULT};  /* declare model instance */
   ```

   The **static** or **auto** keywords can be coded on the declaration in order to give the model instance static or dynamic lifetime, as required.
2. Use the **calloc** or **malloc** functions to obtain storage for a dynamic instance of the structure:

   ```
   PMQMD Instance;
   Instance = malloc(sizeof(MQMD));  /* get storage for dynamic instance */
   ```
3. Use the **memcpy** function to copy the model instance to the dynamic instance:

   ```
   memcpy(Instance,&Model,sizeof(MQMD));  /* initialize dynamic instance */
   ```

Use from C++:

For the C++ programming language, the header files contain the following additional statements that are included only when you use a C++ compiler:

```
#ifdef __cplusplus
  extern "C" {
#endif

/* rest of header file */

#ifdef __cplusplus
  }
#endif
```

Notational conventions:

This information shows how to invoke the functions and declare parameters.

In some cases, the parameters are arrays with a size that is not fixed. For these, a lowercase n is used to represent a numeric constant. When you code the declaration for that parameter, replace the n with the numeric value required.

COBOL programming:

This section contains information to help you use the MQI from the COBOL programming language.

COPY files:

Various COPY files are provided to help you write COBOL application programs that use the MQI. There are two files containing named constants, and two files for each of the structures.

Each structure is provided in two forms: a form with initial values, and a form without:

- Use the structures with initial values in the WORKING-STORAGE SECTION of a COBOL program; they are contained in COPY files with names suffixed with the letter V (for Values).
- Use the structures without initial values in the **LINKAGE SECTION** of a COBOL program; they are contained in COPY files with names suffixed with the letter L (for Linkage).

The COPY files are summarized in Table 118. Not all the files listed are available in all environments.

Table 118. COBOL COPY files

File (with initial values)	File (without initial values)	Contents
CMQAIRV	CMQAIRL	Authentication information record
CMQBOV	CMQBOL	Begin options structure
CMQCIHV	CMQCIHL	CICS information header structure
CMQCNOV	CMQCNOL	Connect options structure
CMQDHV	CMQDHL	Distribution header structure
CMQDLHV	CMQDLHL	Dead letter header structure
CMQDXPV	CMQDXPL	Data conversion exit parameter structure
CMQGMOV	CMQGMOL	Get message options structure
CMQIIHV	CMQIIHL	IMS information header structure
CMQMDV	CMQMDL	Message descriptor structure
CMQMDEV	CMQMDEL	Message descriptor extension structure
CMQMD1V	CMQMD1L	Message descriptor structure version 1
CMQODV	CMQODL	Object descriptor structure
CMQORV	CMQORL	Object record structure
CMQPMOV	CMQPMOL	Put message options structure
CMQRFHV	CMQRFHL	Rules and formatting header structure
CMQRFH2V	CMQRFH2L	Rules and formatting header structure version 2
CMQRMHV	CMQRMHL	Reference message header structure
CMQRRV	CMQRRL	Response record structure
CMQSCOV	CMQSCOL	SSL configuration options

Table 118. COBOL COPY files (continued)

File (with initial values)	File (without initial values)	Contents
CMQTMV	CMQTML	Trigger message structure
CMQTMCV	CMQTMCL	Trigger message structure (character format)
CMQTMC2V	CMQTMC2L	Trigger message structure (character format) version 2
CMQWIHV	CMQWIHL	Work information header structure
CMQXQHV	CMQXQHL	Transmission queue header structure
CMQV	-	Named constants for main MQI
CMQXV	-	Named constants for data conversion exit
CMQMD2V	CMQMD2L	Message descriptor structure version 2

Structures:

In the COPY file, each structure declaration begins with a level-10 item; this enables you to declare several instances of the structure by coding the level-01 declaration and then using the **COPY** statement to copy in the remainder of the structure declaration. To refer to the appropriate instance, use the **IN** keyword:

```
* Declare two instances of MQMD
 01  MY-MQMD.
     COPY CMQMDV.
 01  MY-OTHER-MQMD.
     COPY CMQMDV.
*
* Set MSGTYPE field in MY-OTHER-MQMD
     MOVE MQMT-REQUEST TO MQMD-MSGTYPE IN MY-OTHER-MQMD.
```

Align the structures on appropriate boundaries. If you use the **COPY** statement to include a structure following an item that is not the level-01 item, ensure that the structure begins at the appropriate offset from the start of the level-01 item. Most MQI structures require 4-byte alignment; the exceptions to this are MQCNO, MQOD, and MQPMO, which require 16-byte alignment on IBM i.

In this section, the names of fields in structures are shown without a prefix. In COBOL, the field names are prefixed with the name of the structure followed by a hyphen. However, if the structure name ends with a numeric digit, indicating that the structure is a second or later version of the original structure, the numeric digit is omitted from the prefix. Field names in COBOL are shown in uppercase (although lowercase or mixed case can be used if required). For example, the field *MsgType* described in "MQMD - Message descriptor" on page 1705 becomes MQMD-MSGTYPE in COBOL.

The V-suffix structures are declared with initial values for all the fields; you need to set only those fields where you want a value that is different from the supplied initial value.

Pointers:

Some structures need to address optional data that might be discontiguous with the structure, such as the MQOR and MQRR records addressed by the MQOD structure.

To address this optional data, the structures contain fields that are declared with the pointer data type. However, COBOL does not support the pointer data type in all environments. Because of this, the optional data can also be addressed using fields that contain the offset of the data from the start of the structure.

If you want to port an application between environments, ascertain whether the pointer data type is available in all the intended environments. If it is not, the application must address the optional data using the offset fields instead of the pointer fields.

In those environments where pointers are not supported, declare the pointer fields as byte strings of the appropriate length, with the initial value being the all-null byte string. Do not alter this initial value if you are using the offset fields.

Named constants:

In this section, the names of constants are shown containing the underscore character (_) as part of the name. In COBOL, use the hyphen character (-) in place of the underscore.

Constants that have character-string values use the single quotation mark as the string delimiter ('). In some environments, you might have to specify an appropriate compiler option to cause the compiler to accept the single quotation mark as the string delimiter in place of the double quotation mark.

The named constants are declared in the COPY files as level-10 items. To use the constants, declare the level-01 item explicitly, and then use the **COPY** statement to copy in the declarations of the constants:

```
* Declare a structure to hold the constants
 01  MY-MQ-CONSTANTS.
     COPY CMQV.
```

The preceding method causes the constants to occupy storage in the program even if they are not referenced. If you include the constants in many separate programs within the same run unit, multiple copies of the constants exist, consuming main storage unnecessarily. Avoid this effect by using one of the following techniques:

- Add the **GLOBAL** clause to the level-01 declaration:

  ```
  * Declare a global structure to hold the constants
   01  MY-MQ-CONSTANTS GLOBAL.
       COPY CMQV.
  ```

  This causes allocates storage for only one set of constants within the run unit. The constants, however, can be referenced by any program within the run unit, not just the program that contains the level-01 declaration.

  Note: The **GLOBAL** clause is not supported in all environments.
- Manually copy into each program only those constants that are referenced by that program. Do not use the **COPY** statement to copy all the constants into the program.

Notational conventions:

The latter topics in this section show how to invoke the calls and declare parameters. In some cases, the parameters are tables or character strings the size of which is not fixed. For these, a lowercase n is used to represent a numeric constant. When you code the declaration for that parameter, replace the n with the numeric value required.

System/390 assembler programming:

This section contains information to help to you use the MQI from the System/390 Assembler programming language.

Macros:

Various macros are provided to help you to write assembler application programs that use the MQI.

There are two macros for named constants, and one macro for each of the structures. These files are summarized in Table 119.

Table 119. Assembler macros

File	Contents
CMQA	Named constants (equates) for main MQI
CMQCIHA	CICS information header structure
CMQCNOA	Connect options structure
CMQDLHA	Dead letter header structure
CMQDXPA	Data conversion exit parameter structure
CMQGMOA	Get message options structure
CMQIIHA	IMS information header structure
CMQMDA	Message descriptor structure
CMQMDEA	Message descriptor extension structure
CMQODA	Object descriptor structure
CMQPMOA	Put message options structure
CMQRFHA	Rules and formatting header structure
CMQRFH2A	Rules and formatting header structure version 2
CMQRMHA	Reference message header structure
CMQTMA	Trigger message structure
CMQTMC2A	Trigger message structure (character format) version 2
CMQVERA	Structure version control
CMQWIHA	Work information header structure
CMQXA	Named constants for data conversion exit
CMQXPA	API crossing exit parameter structure
CMQXQHA	Transmission queue header structure

Structures:

The structures are generated by macros that have various parameters to control the action of the macro. These parameters are described in the following sections.

From time to time new versions of the MQ structures are introduced. The additional fields in a new version can cause a structure that previously was smaller than 256 bytes to become larger than 256 bytes. Because of this, write assembler instructions that are intended to copy an MQ structure, or to set an MQ structure to nulls, to work correctly with structures that might be larger than 256 bytes. Alternatively, use the **DCLVER** macro parameter or CMQVERA macro with the **VERSION** parameter to declare a specific version of the structure.

Specifying the name of the structure:

To declare more than one instance of a structure, the macro prefixes the name of each field in the structure with a user-specifiable string and an underscore.

The string used is the label specified on the invocation of the macro. If no label is specified, the name of the structure is used to construct the prefix:

```
* Declare two object descriptors
         CMQODA  ,        Prefix used="MQOD_" (the default)
MY_MQOD  CMQODA  ,        Prefix used="MY_MQOD_"
```

The structure declarations shown in this section use the default prefix.

Specifying the form of the structure:

Structure declarations can be generated by the macro in one of two forms, controlled by the **DSECT** parameter:

DSECT=YES
: An assembler **DSECT** instruction is used to start a new data section; the structure definition immediately follows the **DSECT** statement. The label on the macro invocation is used as the name of the data section; if no label is specified, the name of the structure is used.

DSECT=NO
: Assembler **DC** instructions are used to define the structure at the current position in the routine. The fields are initialized with values, which can be specified by coding the relevant parameters on the macro invocation. Fields for which no values are specified on the macro invocation are initialized with default values.

The value specified must be uppercase. If the **DSECT** parameter is not specified, **DSECT=NO** is assumed.

Controlling the version of the structure:

By default, the macros always declare the most recent version of each structure.

Although you can use the **VERSION** macro parameter to specify a value for the *Version* field in the structure, that parameter defines the initial value for the *Version* field, and does not control the version of the structure actually declared. To control the version of the structure that is declared, use the **DCLVER** parameter:

DCLVER=CURRENT
: The version declared is the current (most recent) version.

DCLVER=SPECIFIED
: The version declared is the version specified by the **VERSION** parameter. If you omit the **VERSION** parameter, the default is version 1.

  If you specify the **VERSION** parameter, the value must be a self-defining numeric constant, or the named constant for the version required (for example, MQCNO_VERSION_3). If you specify some other value, the structure is declared as if **DCLVER=CURRENT** had been specified, even if the value of **VERSION** resolves to a valid value.

The value specified must be uppercase. If you omit the **DCLVER** parameter, the value used is taken from the **MQDCLVER** global macro variable. You can set this variable using the CMQVERA macro.

Declaring one structure embedded within another:

To declare one structure as a component of another structure, use the **NESTED** parameter:

NESTED=YES
The structure declaration is nested within another.

NESTED=NO
The structure declaration is not nested within another.

The value specified must be uppercase. If you omit the **NESTED** parameter, **NESTED=NO** is assumed.

Specifying initial values for fields:

Specify the value to be used to initialize a field in a structure by coding the name of that field (without the prefix) as a parameter on the macro invocation, accompanied by the value required.

For example, to declare a message-descriptor structure with the *MsgType* field initialized with MQMT_REQUEST, and the *ReplyToQ* field initialized with the string "MY_REPLY_TO_QUEUE", use the following:

```
MY_MQMD  CMQMDA  MSGTYPE=MQMT_REQUEST,                          X
               REPLYTOQ=MY_REPLY_TO_QUEUE
```

If you specify a named constant (equate) as a value on the macro invocation, use the CMQA macro to define the named constant. Do not enclose character string values in single quotation marks.

Controlling the listing:

Control the appearance of the structure declaration in the assembler listing using the **LIST** parameter:

LIST=YES
The structure declaration appears in the assembler listing.

LIST=NO
The structure declaration does not appear in the assembler listing.

The value specified must be uppercase. If you omit the **LIST** parameter, **LIST=NO** is assumed.

CMQVERA macro:

This macro allows you to set the default value to be used for the **DCLVER** parameter on the structure macros. The value specified by CMQVERA is used by the structure macro only if you omit the **DCLVER** parameter from the invocation of the structure macro. The default value is set by coding the CMQVERA macro with the **DCLVER** parameter:

DCLVER=CURRENT
The default version is set to the current (most recent) version.

DCLVER=SPECIFIED
The default version is set to the version specified by the **VERSION** parameter.

You must specify the **DCLVER** parameter, and the value must be uppercase. The value set by CMQVERA remains the default value until the next invocation of CMQVERA, or the end of the assembly. If you omit CMQVERA, the default is **DCLVER=CURRENT**.

Notational conventions:

Later sections show how to invoke the calls and declare parameters. In some cases, the parameters are arrays or character strings with a size that is not fixed for which, a lowercase n is used to represent a numeric constant. When you code the declaration for that parameter, replace the n with the numeric value required.

MQAIR - Authentication information record:

The MQAIR structure represents the authentication information record.

The following table summarizes the fields in the structure.

Table 120. Fields in MQAIR

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
AuthInfoType	Type of authentication information	AuthInfoType
AuthInfoConnName	Connection name of LDAP CRL server	AuthInfoConnName
LDAPUserNamePtr	Address of LDAP user name	LDAPUserNamePtr
LDAPUserNameOffset	Offset of LDAP user name from start of MQSCO	LDAPUserNameOffset
LDAPUserNameLength	Length of LDAP user name	LDAPUserNameLength
LDAPPassword	Password to access LDAP server	LDAPPassword
Note: The remaining fields are ignored if *Version* is less than MQAIR_VERSION_2.		
OCSPResponderURL	URL at which the OCSP responder can be contacted	OCSPResponderURL

Overview for MQAIR:

The MQAIR structure allows an application running as a WebSphere MQ MQI client to specify information about an authenticator that is to be used for the client connection. The structure is an input parameter on the MQCONNX call.

Availability: AIX, HP-UX, Solaris, Linux and Windows clients.

Character set and encoding: Data in MQAIR must be in the character set and encoding of the local queue manager; these are given by the **CodedCharSetId** queue manager attribute and MQENC_NATIVE.

Fields for MQAIR:

The MQAIR structure contains the following fields; the fields are described in **alphabetical order**:

AuthInfoConnName (MQCHAR264):

This is either the host name or the network address of a host on which the LDAP server is running. This can be followed by an optional port number, enclosed in parentheses. The default port number is 389.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_CONN_NAME_ERROR.

This is an input field. The length of this field is given by MQ_AUTH_INFO_CONN_NAME_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

AuthInfoType (MQLONG):

This is the type of authentication information contained in the record.

The value can be one of the two following parameters:

MQAIT_CRL_LDAP
Certificate revocation checking using LDAP server.

MQAIT_OCSP
Certificate revocation checking using OCSP.

If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_TYPE_ERROR.

This is an input field. The initial value of this field is MQAIT_CRL_LDAP.

LDAPPassword (MQCHAR32):

This is the password needed to access the LDAP CRL server. If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field.

If the LDAP server does not require a password, or you omit the LDAP user name, *LDAPPassword* must be null or blank. If you omit the LDAP user name and *LDAPPassword* is not null or blank, the call fails with reason code MQRC_LDAP_PASSWORD_ERROR.

This is an input field. The length of this field is given by MQ_LDAP_PASSWORD_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

LDAPUserNameLength (MQLONG):

This is the length in bytes of the LDAP user name addressed by the *LDAPUserNamePtr* or *LDAPUserNameOffset* field. The value must be in the range zero through MQ_DISTINGUISHED_NAME_LENGTH. If the value is not valid, the call fails with reason code MQRC_LDAP_USER_NAME_LENGTH_ERR.

If the LDAP server involved does not require a user name, set this field to zero.

This is an input field. The initial value of this field is 0.

LDAPUserNameOffset (MQLONG):

This is the offset in bytes of the LDAP user name from the start of the MQAIR structure.

The offset can be positive or negative. The field is ignored if *LDAPUserNameLength* is zero.

You can use either *LDAPUserNamePtr* or *LDAPUserNameOffset* to specify the LDAP user name, but not both; see the description of the *LDAPUserNamePtr* field for details.

This is an input field. The initial value of this field is 0.

LDAPUserNamePtr (PMQCHAR):

This is the LDAP user name.

It consists of the Distinguished Name of the user who is attempting to access the LDAP CRL server. If the value is shorter than the length specified by *LDAPUserNameLength*, terminate the value with a null character, or pad it with blanks to the length *LDAPUserNameLength*. The field is ignored if *LDAPUserNameLength* is zero.

You can supply the LDAP user name in one of two ways:

- By using the pointer field *LDAPUserNamePtr*

  In this case, the application can declare a string that is separate from the MQAIR structure, and set *LDAPUserNamePtr* to the address of the string.

  Consider using *LDAPUserNamePtr* for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).
- By using the offset field *LDAPUserNameOffset*

  In this case, the application must declare a compound structure containing the MQSCO structure followed by the array of MQAIR records followed by the LDAP user name strings, and set *LDAPUserNameOffset* to the offset of the appropriate name string from the start of the MQAIR structure. Ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

  Consider using *LDAPUserNameOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that might not be portable to different environments (for example, the COBOL programming language).

Whichever technique is chosen, use only one of *LDAPUserNamePtr* and *LDAPUserNameOffset*; the call fails with reason code MQRC_LDAP_USER_NAME_ERROR if both are nonzero.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

OCSPResponderURL (MQCHAR256):

For an MQAIR structure that represents connection details for an OCSP responder, this field contains the URL at which the responder can be contacted.

The value of this field is an HTTP URL. This field takes priority over a URL in an AuthorityInfoAccess (AIA) certificate extension.

The value is ignored unless both the following statements are true:

- The MQAIR structure is Version 2 or later (the Version field is set to MQAIR_VERSION_2 or greater).
- The AuthInfoType field is set to MQAIT_OCSP.

If the field does not contain an HTTP URL in the correct format (and is not being ignored), the MQCONNX call fails with reason code MQRC_OCSP_URL_ERROR.

This field is case-sensitive. It must start with the string http:// in lower case. The rest of the URL might be case-sensitive, depending on the OCSP server implementation.

This field is not subject to data conversion.

StrucId (MQCHAR4):

The value must be:

MQAIR_STRUC_ID
Identifier for the authentication information record.

For the C programming language, the constant MQAIR_STRUC_ID_ARRAY is also defined; this has the same value as MQAIR_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQAIR_STRUC_ID.

Version (MQLONG):

The version number of the MQAIR structure.

The value must be one of the following:

MQAIR_VERSION_1
Version-1 authentication information record.

MQAIR_VERSION_2
Version-2 authentication information record.

The following constant specifies the version number of the current version:

MQAIR_CURRENT_VERSION
Current version of authentication information record.

This is always an input field. The initial value of this field is MQAIR_VERSION_1.

Initial values and language declarations for MQAIR:

Table 121. Initial values of fields in MQAIR

Field name	Name of constant	Value of constant
StrucId	MQAIR_STRUC_ID	'AIR¬'
Version	MQAIR_VERSION_1	1
AuthInfoType	MQAIT_CRL_LDAP	1
AuthInfoConnName	None	Null string or blanks
LDAPUserNamePtr	None	Null pointer or null bytes
LDAPUserNameOffset	None	0
LDAPUserNameLength	None	0
LDAPPassword	None	Null string or blanks
OCSPResponderURL	None	Null string or blanks

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQAIR_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQAIR MyAIR = {MQAIR_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQAIR MQAIR;
struct tagMQAIR {
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQLONG     AuthInfoType;        /* Type of authentication
                                     information */
  MQCHAR264  AuthInfoConnName;    /* Connection name of CRL LDAP
                                     server */
  PMQCHAR    LDAPUserNamePtr;     /* Address of LDAP user name */
  MQLONG     LDAPUserNameOffset;  /* Offset of LDAP user name from start
                                     of MQAIR structure */
  MQLONG     LDAPUserNameLength;  /* Length of LDAP user name */
  MQCHAR32   LDAPPassword;        /* Password to access LDAP server */
  MQCHAR256  OCSPResponderURL;    /* URL of OCSP responder */

};
```

COBOL declaration:

```
**   MQAIR structure
  10 MQAIR.
**    Structure identifier
   15 MQAIR-STRUCID            PIC X(4).
**    Structure version number
   15 MQAIR-VERSION            PIC S9(9) BINARY.
**    Type of authentication information
   15 MQAIR-AUTHINFOTYPE       PIC S9(9) BINARY.
**    Connection name of CRL LDAP server
   15 MQAIR-AUTHINFOCONNNAME   PIC X(264).
**    Address of LDAP user name
   15 MQAIR-LDAPUSERNAMEPTR    POINTER.
**    Offset of LDAP user name from start of MQAIR structure
   15 MQAIR-LDAPUSERNAMEOFFSET PIC S9(9) BINARY.
**    Length of LDAP user name
   15 MQAIR-LDAPUSERNAMELENGTH PIC S9(9) BINARY.
```

```
**    Password to access LDAP server
   15 MQAIR-LDAPPASSWORD       PIC X(32).
**    URL of OCSP responder
   15 MQAIR-OCSPRESPONDERURL   PIC X(256).
```

Visual Basic declaration:

```
Type MQAIR
  StrucId            As String*4   'Structure identifier'
  Version            As Long       'Structure version number'
  AuthInfoType       As Long       'Type of authentication information'
  AuthInfoConnName   As String*264 'Connection name of CRL LDAP server'
  LDAPUserNamePtr    As MQPTR      'Address of LDAP user name'
  LDAPUserNameOffset As Long       'Offset of LDAP user name from start'
                                   'of MQAIR structure'
  LDAPUserNameLength As Long       'Length of LDAP user name'
  LDAPPassword       As String*32  'Password to access LDAP server'
End Type
```

MQBMHO - Buffer to message handle options:

The following table summarizes the fields in the structure. MQBMHO structure - buffer to message handle options

Table 122. Fields in MQBMHO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options controlling the action of MQBMHO	Options

Overview for MQBMHO:

Availability: All. Buffer to message handle options structure - overview

Purpose: The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers. The structure is an input parameter on the MQBUFMH call.

Character set and encoding: Data in MQBMHO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQBMHO:

Buffer to message handle options structure - fields

The MQBMHO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Buffer to message handle structure - Options field

The value can be:

MQBMHO_DELETE_PROPERTIES
Properties that are added to the message handle are deleted from the buffer. If the call fails no properties are deleted.

Default options: If you do not need the option described, use the following option:

MQBMHO_NONE
No options specified.

This is always an input field. The initial value of this field is MQBMHO_DELETE_PROPERTIES.

StrucId (MQCHAR4):

Buffer to message handle structure - StrucId field

This is the structure identifier. The value must be:

MQBMHO_STRUC_ID
Identifier for buffer to message handle structure.

For the C programming language, the constant MQBMHO_STRUC_ID_ARRAY is also defined; this has the same value as MQBMHO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQBMHO_STRUC_ID.

Version (MQLONG):

Buffer to message handle structure - Version field

This is the structure version number. The value must be:

MQBMHO_VERSION_1
Version number for buffer to message handle structure.

The following constant specifies the version number of the current version:

MQBMHO_CURRENT_VERSION
Current version of buffer to message handle structure.

This is always an input field. The initial value of this field is MQBMHO_VERSION_1.

Initial values and language declarations for MQBMHO:

Buffer to message handle structure - Initial values

Table 123. Initial values of fields in MQBMHO

Field name	Name of constant	Value of constant
StrucId	MQBMHO_STRUC_ID	'BMHO'
Version	MQBMHO_VERSION_1	1
Options	MQBMHO_NONE	0

Notes:

1. In the C programming language, the macro variable MQBMHO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQBMHO MyBMHO = {MQBMHO_DEFAULT};
```

C declaration:

Buffer to message handle structure - C language declaration

```
typedef struct tagMQBMHO MQBMHO;
struct tagMQBMHO {
  MQCHAR4  StrucId;        /* Structure identifier */
  MQLONG   Version;        /* Structure version number */
  MQLONG   Options;        /* Options that control the action of
                              MQBUFMH */
};
```

COBOL declaration:

Buffer to message handle structure - COBOL language declaration

```
**   MQBMHO structure
  10 MQBMHO.
**    Structure identifier
   15 MQBMHO-STRUCID               PIC X(4).
**    Structure version number
   15 MQBMHO-VERSION               PIC S9(9) BINARY.
**    Options that control the action of MQBUFMH
   15 MQBMHO-OPTIONS               PIC S9(9) BINARY.
```

PL/I declaration:

Buffer to message handle structure - PL/I language declaration

```
Dcl
 1 MQBMHO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action
                                     of MQBUFMH */
```

High Level Assembler declaration:

Buffer to message handle structure - Assembler language declaration

```
MQBMHO                  DSECT
MQBMHO_STRUCID          DS   CL4  Structure identifier
MQBMHO_VERSION          DS   F    Structure version number
MQBMHO_OPTIONS          DS   F    Options that control the
*                                  action of MQBUFMH
MQBMHO_LENGTH           EQU  *-MQBMHO
MQBMHO_AREA             DS   CL(MQBMHO_LENGTH)
```

MQBO - Begin options:

The following table summarizes the fields in the structure.

Table 124. Fields in MQBO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options that control the action of MQBEGIN	Options

Overview for MQBO:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows; not available for WebSphere MQ MQI clients.

Purpose: The MQBO structure allows the application to specify options relating to the creation of a unit of work. The structure is an input/output parameter on the MQBEGIN call.

Character set and encoding: Data in MQBO must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQBO:

The MQBO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

This field is always an input field. Its initial value is MQBO_NONE.

The value must be:

MQBO_NONE
No options specified.

StrucId (MQCHAR4):

This field is always an input field. Its initial value is MQBO_STRUC_ID.

The value must be:

MQBO_STRUC_ID
Identifier for begin-options structure.

For the C programming language, the constant MQBO_STRUC_ID_ARRAY is also defined; this has the same value as MQBO_STRUC_ID, but is an array of characters instead of a string.

Version (MQLONG):

This field is always an input field. Its initial value is MQBO_VERSION_1.

The value must be:

MQBO_VERSION_1
Version number for begin-options structure.

The following constant specifies the version number of the current version:

MQBO_CURRENT_VERSION
Current version of begin-options structure.

Initial values and language declarations for MQBO:

Table 125. Initial values of fields in MQBO for MQBO

Field name	Name of constant	Value of constant
StrucId	MQBO_STRUC_ID	'BO¬¬'
Version	MQBO_VERSION_1	1
Options	MQBO_NONE	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQBO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQBO MyBO = {MQBO_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQBO MQBO;
struct tagMQBO {
  MQCHAR4  StrucId;  /* Structure identifier */
  MQLONG   Version;  /* Structure version number */
  MQLONG   Options;  /* Options that control the action of MQBEGIN */
};
```

COBOL declaration:

```
**   MQBO structure
  10 MQBO.
**    Structure identifier
   15 MQBO-STRUCID PIC X(4).
**    Structure version number
   15 MQBO-VERSION PIC S9(9) BINARY.
**    Options that control the action of MQBEGIN
   15 MQBO-OPTIONS PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQBO based,
  3 StrucId char(4),       /* Structure identifier */
  3 Version fixed bin(31), /* Structure version number */
  3 Options fixed bin(31); /* Options that control the action of
                              MQBEGIN */
```

Visual Basic declaration:

```
Type MQBO
  StrucId As String*4 'Structure identifier'
  Version As Long     'Structure version number'
  Options As Long     'Options that control the action of MQBEGIN'
End Type
```

MQCBC - Callback context:

The following table summarizes the fields in the structure. Structure describing the callback routine.

Table 126. Fields in MQCBC

Field	Description	Topic
StrucID	Structure identifier	StrucID
Version	Structure version number	Version
CallType	Why function has been called	CallType
Hobj	Object handle	Hobj
CallbackArea	Field for callback function to use	CallbackArea
ConnectionArea	Field for callback function to use	ConnectionArea
CompCode	Completion code	CompCode
Reason	Reason code	Reason
State	Indication of the state of the current consumer	State
DataLength	Message length	DataLength
BufferLength	Length of message buffer in bytes	BufferLength
Flags	General flags	Flags
Note: The remaining field is ignored if Version is less than MQCBC_VERSION_2		
ReconnectDelay	Number of milliseconds before reconnection attempt	ReconnectDelay

Overview for MQCBC:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS, plus WebSphere MQ MQI clients connected to these systems.

Purpose: The MQCBC structure is used to specify context information that is passed to a callback function.

The structure is an input/output parameter on the call to a message consumer routine.

Version: The current version of MQCBC is MQCBC_VERSION_2.

Character set and encoding: Data in MQCBC must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure will be in the character set and encoding of the client.

Fields for MQCBC:

Alphabetic list of fields for the MQCBC structure.

The MQCBC structure contains the following fields; the fields are described in alphabetical order:

BufferLength (MQLONG):

This field is the length in bytes of the message buffer that has been passed to this function.

The buffer can be larger than both the MaxMsgLength value defined for the consumer and the ReturnedLength value in the MQGMO.

The actual message length is supplied in DataLength field.

The application can use the entire buffer for its own purposes for the duration of the callback function.

This is an input field to the message consumer function; it is not relevant to an exception handler function.

CallbackArea (MQPTR):

This field is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the CallbackArea field in the MQCBD structure, which is a parameter on the MQCB call used to define the callback function.

Changes to the *CallbackArea* are preserved across the invocations of the callback function for an *HObj*. This field is not shared with callback functions for other handles.

This is an input/output field to the callback function. The initial value of this field is a null pointer or null bytes.

CallType (MQLONG):

Field containing information about why this function has been called; the following are defined.

Message delivery call types: These call types contain information about a message. The *DataLength* and *BufferLength* parameters are valid for these call types.

MQCBCT_MSG_REMOVED
The message consumer function has been invoked with a message that has been destructively removed from the object handle.

If the value of *CompCode* is MQCC_WARNING, the value of the *Reason* field is MQRC_TRUNCATED_MSG_ACCEPTED or one of the codes indicating a data conversion problem.

MQCBCT_MSG_NOT_REMOVED
The message consumer function has been invoked with a message that has not yet been destructively removed from the object handle. The message can be destructively removed from the object handle using the *MsgToken*.

The message might not have been removed because:

- The MQGMO options requested a browse operation, MQGMO_BROWSE_*
- The message is larger than the available buffer and the MQGMO options do not specify MQGMO_ACCEPT_TRUNCATED_MSG

If the value of *CompCode* is MQCC_WARNING, the value of the *Reason* field is MQRC_TRUNCATED_MSG_FAILED or one of the codes indicating a data conversion problem.

Callback control call types: These call types contain information about the control of the callback and do not contain details about a message. These call types are requested using Options in the MQCBD structure.

The *DataLength* and *BufferLength* parameters are not valid for these call types.

MQCBCT_REGISTER_CALL
The purpose of this call type is to allow the callback function to perform some initial setup.

The callback function is invoked is immediately after the callback is registered, that is, upon return from an MQCB call using a value for the *Operation* field of MQOP_REGISTER.

This call type is used both for message consumers and event handlers.

If requested, this is the first invocation of the callback function.

The value of the *Reason* field is MQRC_NONE.

MQCBCT_START_CALL
The purpose of this call type is to allow the callback function to perform some setup when it is started, for example, reinstating resources that were cleaned up when it was previously stopped.

The callback function is invoked when the connection is started using either MQOP_START or MQOP_START_WAIT.

If a callback function is registered within another callback function, this call type is invoked when the callback returns.

This call type is used for message consumers only.

The value of the *Reason* field is MQRC_NONE.

MQCBCT_STOP_CALL
The purpose of this call type is to allow the callback function to perform some cleanup when it is stopped for a while, for example, cleaning up additional resources that have been acquired during the consuming of messages.

The callback function is invoked when an MQCTL call is issued using a value for the *Operation* field of MQOP_STOP.

This call type is used for message consumers only.

The value of the *Reason* field is set to indicate the reason for stopping.

MQCBCT_DEREGISTER_CALL
The purpose of this call type is to allow the callback function to perform final cleanup at the end of the consume process. The callback function is invoked when the:

- Callback function is deregistered using an MQCB call with MQOP_DEREGISTER.
- Queue is closed, causing an implicit deregister. In this instance the callback function is passed MQHO_UNUSABLE_HOBJ as the object handle.
- MQDISC call completes - causing an implicit close and, therefore, a deregister. In this case the connection is not disconnected immediately, and any ongoing transaction is not yet committed.

If any of these actions are taken inside the callback function itself, the action is invoked once the callback returns.

This call type is used both for message consumers and event handlers.

If requested, this is the last invocation of the callback function.

The value of the *Reason* field is set to indicate the reason for stopping.

MQCBCT_EVENT_CALL
Event handler function

The event handler function has been invoked without a message when the queue manager or connection stops or quiesces.

This call can be used to take appropriate action for all callback functions.**Message consumer function**

The message consumer function has been invoked without a message when an error (*CompCode*= MQCC_FAILED) has been detected that is specific to the object handle; for example *Reason* code = MQRC_GET_INHIBITED.

The value of the *Reason* field is set to indicate the reason for the call.

MQCBCT_MC_EVENT_CALL

The event handler function has been invoked for multicast events; The event handler is sent WebSphere MQ Multicast events instead of 'normal' WebSphere MQ events.

For more information about MQCBCT_MC_EVENT_CALL, see Multicast exception reporting .

CompCode (MQLONG):

This field is the completion code. It indicates whether there were any problems consuming the message.

The value is one of the following:

MQCC_OK
Successful completion

MQCC_WARNING
Warning (partial completion)

MQCC_FAILED
Call failed

This is an input field. The initial value of this field is MQCC_OK.

ConnectionArea (MQPTR):

This field is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the ConnectionArea field in the MQCTLO structure, which is a parameter on the MQCTL call used to control the callback function.

Any changes made to this field by the callback functions are preserved across the invocations of the callback function. This area can be used to pass information that is to be shared by all callback functions. Unlike *CallbackArea*, this area is common across all callbacks for a connection handle.

This is an input and output field. The initial value of this field is a null pointer or null bytes.

DataLength (MQLONG):

This is the length in bytes of the application data in the message. If the value is zero, it means that the message contains no application data.

The DataLength field contains the length of the message but not necessarily the length of the message data passed to the consumer. It could be that the message was truncated. Use the ReturnedLength field in the MQGMO to determine how much data has actually been passed to the consumer.

If the reason code indicates the message has been truncated, you can use the DataLength field to determine how large the actual message is. This allows you to determine the size of the buffer required to accommodate the message data, and then issue an MQCB call to update the MaxMsgLength with an appropriate value.

If the MQGMO_CONVERT option is specified, the converted message could be larger than the value returned for DataLength. In such cases, the application probably needs to issue an MQCB call to update the MaxMsgLength to to be greater than the value returned by the queue manager for DataLength.

To avoid message truncation problems, specify MaxMsgLength as MQCBD_FULL_MSG_LENGTH. This causes the queue manager to allocate a buffer for the full message length after data conversion. Be aware, however, that even if this option is specified, it is still possible that sufficient storage is not available to correctly process the request. Applications should always check the returned reason code. For example, if it is not possible to allocate sufficient storage to convert the message, the messages is returned to the application unconverted.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Flags (MQLONG):

Flags containing information about this consumer.

The following option is defined:

MQCBCF_READA_BUFFER_EMPTY
This flag can be returned if a previous MQCLOSE call using the MQCO_QUIESCE option failed with a reason code of MQRC_READ_AHEAD_MSGS.

This code indicated that the last read ahead message is being returned and that the buffer is now empty. If the application issues another MQCLOSE call using the MQCO_QUIESCE) option, it succeeds.

Note, that an application is not guaranteed to be given a message with this flag set, as there might still be messages in the read-ahead buffer that do not match the current selection criteria. In this instance, the consumer function is invoked with the reason code MQRC_HOBJ_QUIESCED.

If the read ahead buffer is completely empty, the consumer is invoked with the MQCBCF_READA_BUFFER_EMPTY flag and the reason code MQRC_HOBJ_QUIESCED_NO_MSGS.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Hobj (MQHOBJ):

This is the object handle for calls to the message consumer.

For an event handler, this value is MQHO_NONE

The application can use this handle and the message token in the Get Message Options block to get the message if a message has not been removed from the queue.

This is always an input field. The initial value of this field is MQHO_UNUSABLE_HOBJ

Reason (MQLONG):

This is the reason code qualifying the *CompCode*.

This is an input field. The initial value of this field is MQRC_NONE.

State (MQLONG):

An indication as to the state of the current consumer. This field is of most value to an application when a nonzero reason code is passed to the consumer function.

You can use this field to simplify application programming because you do not need to code behavior for each reason code.

This is an input field. The initial value of this field is MQCS_NONE

State	Queue manager action	Value of constant
MQCS_NONE This reason code represents a normal call with no additional reason information	None; this is the normal operation.	0
MQCS_SUSPENDED_TEMPORARY These reason codes represent temporary conditions.	The callback routine is called to report the condition and then suspended. After a period of time the system might attempt the operation again, which can lead to the same condition being raised again.	1
MQCS_SUSPENDED_USER_ACTION These reason codes represent conditions where the callback needs to take action to resolve the condition.	The consumer is suspended and the callback routine is called to report the condition. The callback routine should resolve the condition if possible and either RESUME or close down the connection.	2
MQCS_SUSPENDED These reason codes represent failures that prevent further message callbacks.	The queue manager automatically suspends the callback function. If the callback function is resumed it is likely to receive the same reason code again.	3
MQCS_STOPPED These reason codes represent the end of message consumption.	Delivered to the exception handler and to callbacks that specified MQCBDO_STOP_CALL. No further messages can be consumed.	4

StrucId (MQCHAR4):

The value in this field is the structure identifier.

The value must be:

MQCBC_STRUC_ID
Identifier for callback context structure.

For the C programming language, the constant MQCBC_STRUC_ID_ARRAY is also defined; this has the same value as MQCBC_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCBC_STRUC_ID.

Version (MQLONG):

The value in this field is the structure version number.

The value must be:

MQCBC_VERSION_1
Version-1 callback context structure.

The following constant specifies the version number of the current version:

MQCBC_CURRENT_VERSION
Current version of the callback context structure.

This is always an input field. The initial value of this field is MQCBC_VERSION_1.

The callback function is always passed the latest version of the structure.

ReconnectDelay (MQLONG):

ReconnectDelay indicates how long the queue manager will wait before trying to reconnect. The field can be modified by an event handler to change the delay or stop reconnection altogether.

Use the `ReconnectDelay` field only if the value of the `Reason` field in the Callback Context is `MQRC_RECONNECTING`.

On entry to the event handler the value of `ReconnectDelay` is the number of milliseconds the queue manager is going to wait before making a reconnection attempt. Table 127 lists the values that you can set to modify the behavior of the queue manager on return from the event handler.

Table 127. `ReconnectDelay` values

Name	Value	Description
`MQRD_NO_RECONNECT`	`-1`	Make no more reconnection attempts. An error is returned to the application.
`MQRD_NO_DELAY`	`0`	Try to reconnect immediately.
`Milliseconds`	`>0`	Wait for this many milliseconds before retrying the connection.

Initial values and language declarations for MQCBC:

Callback context structure - initial values

There are no initial values for the **MQCBC** structure. The structure is passed as a parameter to a callback routine. The queue manager initializes the structure; applications never initialize it.

C declaration:

Callback context structure - C language declaration

```
typedef struct tagMQCBC MQCBC;
struct tagMQCBC {
   MQCHAR4   StrucId;                /* Structure identifier */
   MQLONG    Version;                /* Structure version number */
   MQLONG    CallType;               /* Why Function was called */
   MQHOBJ    Hobj;                   /* Object Handle */
   MQPTR     CallbackArea;           /* Callback data passed to the function */
   MQPTR     ConnectionArea;         /* MQCTL data area passed to the function */
   MQLONG    CompCode;               /* Completion Code */
   MQLONG    Reason;                 /* Reason Code */
```

```
   MQLONG    State;                  /* Consumer State */
   MQLONG    DataLength;             /* Message Data Length */
   MQLONG    BufferLength;           /* Buffer Length */
   MQLONG    Flags;                  /* Flags containing information about
                                        this consumer */
  /* Ver:1 */
   MQLONG    ReconnectDelay;         /* Number of milliseconds before */
  /* Ver:2 */  };                   /* reconnect attempt */
```

COBOL declaration:

```
** MQCBC structure
 10  MQCBC.
** Structure Identifier
 15  MQCBC-STRUCID                     PIC X(4).
** Structure Version
 15  MQCBC-VERSION                     PIC S9(9) BINARY.
 ** Call Type
 15  MQCBC-CALLTYPE                    PIC S9(9) BINARY.
** Object Handle
 15  MQCBC-HOBJ                        PIC S9(9) BINARY.
** Callback User Area
 15  MQCBC-CALLBACKAREA                POINTER
** Connection Area
 15  MQCBC-CONNECTIONAREA              POINTER
** Completion Code
 15  MQCBC-COMPCODE                    PIC S9(9) BINARY.
** Reason Code
 15  MQCBC-REASON                      PIC S9(9) BINARY.
** Consumer State
 15  MQCBC-STATE                       PIC S9(9) BINARY.
** Data Length
 15  MQCBC-DATALENGTH                  PIC S9(9) BINARY.
** Buffer Length
 15  MQCBC-BUFFERLENGTH                PIC S9(9) BINARY.
** Flags
 15  MQCBC-FLAGS                       PIC S9(9) BINARY.
** Ver:1 **
** Number of milliseconds before reconnect attempt
 15  MQCBC-RECONNECTDELAY PIC S9(9) BINARY.
** Ver:2 **
```

PL/I declaration:

```
dcl
 1 MQCBC based,
  3 StrucId             char(4),       /* Structure identifier */
  3 Version             fixed bin(31), /* Structure version */
  3 CallType            fixed bin(31), /* Callback type */
  3 Hobj                fixed bin(31), /* Object Handle */
  3 CallbackArea        pointer,       /* User area passed to the function */
  3 ConnectionArea      pointer,       /* Connection User Area */
  3 CompCode            fixed bin(31); /* Completion Code */
  3 Reason              fixed bin(31); /* Reason Code */
  3 State               fixed bin(31); /* Consumer State */
  3 DataLength          fixed bin(31); /* Message Data Length */
  3 BufferLength        fixed bin(31); /* Message Buffer length */
  3 Flags               fixed bin(31); /* Consumer Flags */
/* Ver:1 */
  3 ReconnectDelay      fixed bin(31); /* Number of milliseconds before */
/* Ver:2 */                           /* reconnect attempt */
```

High Level Assembler declaration:

```
MQCBC                DSECT
MQCBC                DS  0F    Force fullword alignment
MQCBC_STRUCID        DS  CL4   Structure identifier
MQCBC_VERSION        DS  F     Structure version number
MQCBC_CALLTYPE       DS  F     Why Function was called
MQCBC_HOBJ           DS  F     Object Handle
MQCBC_CALLBACKAREA   DS  A     Callback data passed to the function
MQCBC_CONNECTIONAREA DS  A     MQCTL Data area passed to the function
MQCBC_COMPCODE       DS  F     Completion Code
MQCBC_REASON         DS  F     Reason Code
MQCBC_STATE          DS  F     Consumer State
MQCBC_DATALENGTH     DS  F     Message Data Length
MQCBC_BUFFERLENGTH   DS  F     Buffer Length
MQCBC_FLAGS          DS  F     Flags containing information about this consumer
MQCBC_RECONNECTDELAY DS  F     Number of milliseconds before reconnect
MQCBC_LENGTH         EQU *-MQCBC
                     ORG      MQCBC
MQCBC_AREA           DS  CL(MQCBC_LENGTH)
```

MQCBD - Callback descriptor:

The following table summarizes the fields in the structure. Structure specifying the callback function.

Table 128. Fields in MQCBD

Field	Description	Topic
StrucID	Structure identifier	StrucID
Version	Structure version number	Version
CallbackType	Type of callback function	CallbackType
Options	Options controlling message consumption	Options
Callback Area	Field for callback function to use	CallbackArea
CallbackFunction	Whether the function is invoked as an API call	CallbackFunction
CallbackName	Whether the function is invoked as a dynamically-linked program	CallbackName
MaxMsgLength	Length of longest message that can be read	MaxMsgLength

Overview for MQCBD:
Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS, and WebSphere MQ MQI clients connected to these systems.

Purpose: The MQCBD structure is used to specify a callback function and the options controlling its use by the queue manager.

The structure is an input parameter on the MQCB call.

Version: The current version of MQCBD is MQCBD_VERSION_1.

Character set and encoding: Data in MQCBD must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQCBD:

Alphabetic list of fields for the MQCBD structure.

The MQCBD structure contains the following fields; the fields are described in alphabetical order:

CallbackArea (MQPTR):

Callback descriptor structure - CallbackArea field

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged from the CallbackArea field in the MQCBC structure, which is a parameter on the callback function declaration.

The value is used only on an *Operation* having a value MQOP_REGISTER, with no currently defined callback, it does not replace a previous definition.

This is an input and output field to the callback function. The initial value of this field is a null pointer or null bytes.

CallbackFunction (MQPTR):

Callback descriptor structure - CallbackFunction field

The callback function is invoked as a function call.

Use this field to specify a pointer to the callback function.

You *must* specify either *CallbackFunction* or *CallbackName*. If you specify both, the reason code MQRC_CALLBACK_ROUTINE_ERROR is returned.

If neither *CallbackName* nor *CallbackFunction* is set, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

This option is not supported in the following environment: Programming languages and compilers that do not support function-pointer references. In such situations, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

On z/OS the function must expect to be called with OS linkage conventions. For example, in the C programming language, specify:

```
#pragma linkage(MQCB_FUNCTION,OS)
```

This is an input field. The initial value of this field is a null pointer or null bytes.

Note: When using CICS with WebSphere MQ V7.0.1, asynchronous consumption is supported if:

- Apar PK66866 is applied to CICS TS 3.2
- Apar PK89844 is applied to CICS TS 4.1

CallbackName (MQCHAR128):

Callback descriptor structure - CallbackName field

The callback function is invoked as a dynamically linked program.

You *must* specify either *CallbackFunction* or *CallbackName*. If you specify both, the reason code MQRC_CALLBACK_ROUTINE_ERROR is returned.

If neither *CallbackName* nor *CallbackFunction* is not set, the call fails with the reason code MQRC_CALLBACK_ROUTINE_ERROR.

The module is loaded when the first callback routine to use is registered, and unloaded when the last callback routine to use it deregisters.

Except where noted in the following text, the name is left-justified within the field, with no embedded blanks; the name itself is padded with blanks to the length of the field. In the descriptions that follow, square brackets ([ ]) denote optional information:

IBM i The callback name can be one of the following formats:

- Library "/" Program
- Library "/" ServiceProgram "("FunctionName")"

For example, `MyLibrary/MyProgram(MyFunction)`.

The library name can be *LIBL. Both the library and program names are limited to a maximum of 10 characters.

UNIX systems

The callback name is the name of a dynamically-loadable module or library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path:

```
[path]library(function)
```

If the path is not specified the system search path is used.

The name is limited to a maximum of 128 characters.

Windows

The callback name is the name of a dynamic-link library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses .The library name can optionally be prefixed with a directory path and drive:

```
[d:][path]library(function)
```

If the drive and path are not specified the system search path is used.

The name is limited to a maximum of 128 characters.

z/OS The callback name is the name of a load module that is valid for specification on the EP parameter of the LINK or LOAD macro.

The name is limited to a maximum of 8 characters.

z/OS CICS

The callback name is the name of a load module that is valid for specification on the PROGRAM parameter of the EXEC CICS LINK command macro.

The name is limited to a maximum of 8 characters.

The program can be defined as remote using the REMOTESYTEM option of the installed PROGRAM definition or by the dynamic routing program.

The remote CICS region must be connected to WebSphere MQ if the program is to use WebSphere MQ API calls. Note, however, that the Hobj field in the MQCBC structure is not valid in a remote system.

If a failure occurs trying to load *CallbackName*, one of the following error codes is returned to the application:

- MQRC_MODULE_NOT_FOUND
- MQRC_MODULE_INVALID
- MQRC_MODULE_ENTRY_NOT_FOUND

A message is also written to the error log containing the name of the module for which the load was attempted, and the failing reason code from the operating system.

This is an input field. The initial value of this field is a null string or blanks.

CallbackType (MQLONG):

Callback descriptor structure - CallbackType field

This is the type of the callback function. The value must be one of:

MQCBT_MESSAGE_CONSUMER
Defines this callback as a message consumer function.

A message consumer callback function is called when a message, meeting the selection criteria specified, is available on an object handle and the connection is started.

MQCBT_EVENT_HANDLER
Defines this callback as the asynchronous event routine; it is not driven to consume messages for a handle.

Hobj is not required on the MQCB call defining the event handler and is ignored if specified.

The event handler is called for conditions that affect the whole message consumer environment. The consumer function is invoked without a message when an event, for example, a queue manager or connection stopping, or quiescing, occurs. It is not called for conditions that are specific to a single message consumer, for example, MQRC_GET_INHIBITED.

Events are delivered to the application, regardless of whether the connection is started or stopped, except in the following environments:

- CICS on z/OS environment
- nonthreaded applications

If the caller does not pass one of these values, the call fails with a *Reason* code of MQRC_CALLBACK_TYPE_ERROR

This is always an input field. The initial value of this field is MQCBT_MESSAGE_CONSUMER.

MaxMsgLength (MQLONG):

This is the length in bytes of the longest message that can be read from the handle and given to the callback routine. Callback descriptor structure - MaxMsgLength field

If a message has a longer length, the callback routine receives *MaxMsgLength* bytes of the message, and reason code:

- MQRC_TRUNCATED_MSG_FAILED or
- MQRC_TRUNCATED_MSG_ACCEPTED if you specified MQGMO_ACCEPT_TRUNCATED_MSG.

The actual message length is supplied in the DataLength field of the MQCBC structure.

The following special value is defined:

MQCBD_FULL_MSG_LENGTH
The buffer length is adjusted by the system to return messages without truncation.

If insufficient memory is available to allocate a buffer to receive the message, the system calls the callback function with an MQRC_STORAGE_NOT_AVAILABLE reason code.

If, for example, you request data conversion, and there is insufficient memory available to convert the message data, the unconverted message is passed to the callback function.

This is an input field. The initial value of the *MaxMsgLength* field is MQCBD_FULL_MSG_LENGTH.

Options (MQLONG):

Callback descriptor structure - Options field

Any one, or all, of the following can be specified. If more than one option is required the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQCBDO_FAIL_IF_QUIESCING
The MQCB call fails if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQCB call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

Specify MQGMO_FAIL_IF_QUIESCING, in the MQGMO options passed on the MQCB call, to cause notification to message consumers when they are quiescing.

Control options: The following options control whether the callback function is called, without a message, when the state of the consumer changes:

MQCBDO_REGISTER_CALL
The callback function is invoked with call type MQCBCT_REGISTER_CALL.

MQCBDO_START_CALL
The callback function is invoked with call type MQCBCT_START_CALL.

MQCBDO_STOP_CALL
The callback function is invoked with call type MQCBCT_STOP_CALL.

MQCBDO_DEREGISTER_CALL
The callback function is invoked with call type MQCBCT_DEREGISTER_CALL.

MQCBDO_EVENT_CALL
The callback function is invoked with call type MQCBCT_EVENT_CALL.

MQCBDO_MC_EVENT_CALL
The callback function is invoked with call type MQCBCT__MC_EVENT_CALL.

See CallType for further details about these call types.

Default option: If you do not need any of the options described, use the following option:

MQCBDO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values.

MQCBDO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is MQCBDO_NONE.

StrucId (MQCHAR4):

Callback descriptor structure - StrucId field

This is the structure identifier; the value must be:

MQCBD_STRUC_ID
Identifier for callback descriptor structure.

For the C programming language, the constant MQCBD_STRUC_ID_ARRAY is also defined; this has the same value as MQCBD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCBD_STRUC_ID.

Version (MQLONG):

Callback descriptor structure - Version field

This is the structure version number; the value must be:

MQCBD_VERSION_1
Version-1 callback descriptor structure.

The following constant specifies the version number of the current version:

MQCBD_CURRENT_VERSION
Current version of callback descriptor structure.

This is always an input field. The initial value of this field is MQCBD_VERSION_1.

Initial values and language declarations for MQCBD:

Callback descriptor structure - Initial values

Table 129. Initial values of fields in MQCBD

Field name	Name of constant	Value of constant
StrucId	MQCBD_STRUC_ID	'CBD¬'
Version	MQCBD_VERSION_1	1
CallBackType	MQCBT_MESSAGE_CONSUMER	1
Options	MQCBDO_NONE	0
CallbackArea	None	Null pointer or null blanks
CallbackFunction	None	Null pointer or null blanks
CallbackName	None	Null string or blanks
MaxMsgLength	MQCBD_FULL_MSG_LENGTH	-1

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null sting in the C programming language, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQCBD_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQCBD MyCBD = {MQCBD_DEFAULT};
```

C declaration:

Callback descriptor structure - C language declaration

```
typedef struct tagMQCBD MQCBD;
 struct tagMQCBD {
   MQCHAR4    StrucId;                /* Structure identifier */
   MQLONG     Version;                /* Structure version number */
   MQLONG     CallBackType;           /* Callback function type */
   MQLONG     Options;                /* Options controlling message
                                         consumption */
   MQPTR      CallbackArea;           /* User data passed to the function */
   MQPTR      CallbackFunction;       /* Callback function pointer */
   MQCHAR128  CallbackName;           /* Callback name */
   MQLONG     MaxMsgLength;           /* Maximum message length */
 };
```

COBOL declaration:

```
** MQCBCD structure
 10  MQCBD.
** Structure Identifier
 15  MQCBD-STRUCID                     PIC X(4).
** Structure Version
 15  MQCBD-VERSION                     PIC S9(9) BINARY.
 ** Callback Type
 15  MQCBD-CALLBACKTYPE                PIC S9(9) BINARY.
** Options
 15  MQCBD-OPTIONS                     PIC S9(9) BINARY.
** Callback User Area
 15  MQCBD-CALLBACKAREA                POINTER
** Callback Function Pointer
 15  MQCBD-CALLBACKFUNCTION            FUNCTION-POINTER
** Callback Program Name
 15  MQCBD-CALLBACKNAME                PIC X(128)
** Maximum Message Length
 15  MQCDB-MAXMSGLENGTH                PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQCBD based,
  3 StrucId             char(4),       /* Structure identifier*/
  3 Version             fixed bin(31), /* Structure version*/
  3 CallbackType        fixed bin(31), /* Callback function type */
  3 Options             fixed bin(31), /* Options */
  3 CallbackArea        pointer,       /* User area passed to the function */
  3 CallbackFunction    pointer,       /* Callback Function Pointer */
  3 CallbackName        char(128),     /* Callback Program Name */
  3 MaxMsgLength        fixed bin(31); /* Maximum Message Length */
```

MQCHARV - Variable Length String:

The following table summarizes the fields in the structure.

Field	Description	Topic
VSPtr	Pointer to the variable length string	VSPtr
VSOffset	Offset in bytes of the variable length string from the start of the structure that contains this MQCHARV structure	VSOffset
VSLength	The length in bytes of the variable length string addressed by the VSPtr or VSOffset field.	VSLength
VSBufSize	The size in bytes of the buffer addressed by the VSPtr or VSOffset field.	VSBufSize
VSCCSID	The character set identifier of the variable length string addressed by the VSPtr or VSOffset field.	VSCCSID

Overview for MQCHARV:

Availability: AIX, HP-UX, Solaris, Linux, IBM i, Windows, plus WebSphere MQ MQI clients connected to these systems.

Purpose: Use the MQCHARV structure to describe a variable length string.

Character set and encoding: Data in the MQCHARV must be in the encoding of the local queue manager that is given by MQENC_NATIVE and the character set of the VSCCSID field within the structure. If the application is running as an MQ client, the structure must be in the encoding of the client. Some character sets have a representation that depends on the encoding. If VSCCSID is one of these character sets, the encoding used is the same encoding as that of the other fields in the MQCHARV. The character set identified by VSCCSID can be a double-byte character set (DBCS).

Usage: The MQCHARV structure addresses data that might be discontiguous with the structure containing it. To address this data, fields declared with the pointer data type can be used. Be aware that COBOL does not support the pointer data type in all environments. Because of this, the data can also be addressed using fields that contain the offset of the data from the start of the structure containing the MQCHARV.

COBOL programming

If you want to port an application between environments, you must ascertain whether the pointer data type is available in all the intended environments. If not, the application must address the data using the offset fields instead of the pointer fields.

In those environments where pointers are not supported, you can declare the pointer fields as byte strings of the appropriate length, with the initial value being the all-null byte string. Do not alter this initial value if you are using the offset fields. One way to do this without changing the supplied copy books is to use the following:

```
COPY CMQCHRVV REPLACING POINTER BY ==BINARY PIC S9(9)==.
```

where `CMQCHRVV` can be exchanged for the copy book to be used.

Fields for MQCHARV:

The MQCHARV structure contains the following fields; the fields are described in **alphabetical order**:

VSBufSize (MQLONG):

This is the size in bytes of the buffer addressed by the VSPtr or VSOffset field.

When the MQCHARV structure is used as an output field on a function call, this field must be initialised with the length of the buffer provided. If the value of VSLength is greater than VSBufSize then only VSBufSize bytes of data are returned to the caller in the buffer.

This value must be a value greater than or equal to zero, or the following special value which is recognized:

MQVS_USE_VSLENGTH
When specified, the length of the buffer is taken from the VSLength field in the MQCHARV structure. Do not use this value when using the structure as an output field and a buffer is provided.

This is the initial value of this field.

VSCCSID (MQLONG):

This is the character set identifier of the variable length string addressed by the VSPtr or VSOffset field.

The initial value of this field is MQCCSI_APPL which is defined by MQ to indicate that it should be changed to the true character set identifier of the current process. As a result, the value MQCCSI_APPL is never associated with a variable length string. The initial value of this field can be changed by defining a different value for the constant MQCCSI_APPL for your compile unit by the appropriate means for your application's programming language.

VSLength (MQLONG):

The length in bytes of the variable length string addressed by the VSPtr or VSOffset field.

The initial value of this field is 0. The value must be either greater than or equal to zero or the following special value which is recognized:

MQVS_NULL_TERMINATED

If MQVS_NULL_TERMINATED is not specified, VSLength bytes are included as part of the string. If null characters are present they do not delimit the string.

If MQVS_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null itself is not included as part of that string.

Note: The null character used to terminate a string if MQVS_NULL_TERMINATED is specified is a null from the codeset specified by VSCCSID.

For example, in UTF-16 (UCS-2 CCSIDs 1200 and 13488), this is the two byte Unicode encoding where a null is represented by a 16-bit number of all zeros. In UTF-16 it is common to find single bytes set to all zero which are part of characters (7-bit ASCII characters for instance), but the strings will only be null terminated when two 'zero' bytes are found on an even byte boundary. It is possible to get two 'zero' bytes on an odd boundary when they are each part of valid characters. For example x'01' x'00 x'00' x'30' represents two valid Unicode characters and does not null terminate the string.

VSOffset (MQLONG):

The offset can be positive or negative. You can use either the VSPtr or VSOffset field to specify the variable length string, but not both. The offset in bytes of the variable length string from the start of the MQCHARV, or the structure containing it.

When the MQCHARV structure is embedded within another structure, this value is the offset in bytes of the variable length string from the start of the structure that contains this MQCHARV structure. When the MQCHARV structure is not embedded within another structure, for example, if it is specified as a parameter on a function call, the offset is relative to the start of the MQCHARV structure.

The initial value of this field is 0.

VSPtr (MQPTR):

This is a pointer to the variable length string.

You can use either the VSPtr or VSOffset field to specify the variable length string, but not both.

The initial value of this field is a null pointer or null bytes.

Initial values and language declarations for MQCHARV:

Initial values of fields in MQCHARV

Field name	Name of constant	Value of constant
VSPtr	None	Null pointer or null bytes.
VSOffset	None	0
VSBufSize	MQVS_USE_VSLENGTH	0
VSLength	None	0
VSCCSID	MQCCSI_APPL	-3
Note: In the C programming language, the macro variable MQCHARV_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: `MQCHARV MyVarStr = {MQCHARV_DEFAULT};`		

C declaration:

```
typedef struct tagMQCHARV MQCHARV;
struct tagMQCHARV {
  MQPTR     VSPtr;               /* Address of variable length string */
  MQLONG    VSOffset;            /* Offset of variable length string */
  MQLONG    VSBufSize;           /* Size of buffer */
  MQLONG    VSLength;            /* Length of variable length string */
  MQLONG    VSCCSID;             /* CCSID of variable length string */
};
```

COBOL declaration for MQCHARV:

```
** MQCHARV structure
 10  MQCHARV.
** Address of variable length string
 15  MQCHARV-VSPTR        POINTER.
** Offset of variable length string
 15  MQCHARV-VSOFFSET     PIC S9(9) BINARY.
** Size of buffer
 15  MQCHARV-VSBUFSIZE    PIC S9(9) BINARY.
** Length of variable length string
 15  MQCHARV-VSLENGTH     PIC S9(9) BINARY.
** CCSID of variable length string
 15  MQCHARV-VSCCSID      PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQCHARV based,
  3 VSPtr       pointer,       /* Address of variable length string */
  3 VSOffset    fixed bin(31), /* Offset of variable length string */
  3 VSBufSize   fixed bin(31), /* Size of buffer */
  3 VSLength    fixed bin(31), /* Length of variable length string */
  3 VSCCSID     fixed bin(31); /* CCSID of variable length string */
```

High Level Assembler declaration:

```
MQCHARV             DSECT
MQCHARV_VSPTR       DS   F     Address of variable length string
MQCHARV_VSOFFSET    DS   F     Offset of variable length string
MQCHARV_VSBUFSIZE   DS   F     Size of buffer
MQCHARV_VSLENGTH    DS   F     Length of variable length string
MQCHARV_VSCCSID     DS   F     CCSID of variable length string
*
MQCHARV_LENGTH      EQU  *-MQCHARV
                    ORG  MQCHARV
MQCHARV_AREA        DS   CL(MQCHARV_LENGTH)
```

Redefinition of MQCCSI_APPL:

The following examples show how you can override the value of MQCCSI_APPL in various programming languages. You can change the value of MQCCSI_APPL, removing the need to set the VSCCSID for each variable length string separately.

In these examples the CCSID is set to 1208; change this to the value you require. This becomes the default value, which you can override by setting the VSCCSID in any specific instance of MQCHARV.

C usage

```
#define MQCCSI_APPL 1208
#include <cmqc.h>
```

COBOL usage

```
COPY CMQXYZV REPLACING -3 BY 1208.
```

PL/I usage

```
%MQCCSI_APPL = '1208';
%include syslib(cmqp);
```

System/390 assembler usage

```
MQCCSI_APPL EQU 1208
         CMQA LIST=NO
```

MQCIH - CICS bridge header:

The following table summarizes the fields in the structure.

Table 130. Fields in MQCIH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQCIH structure	StrucLength
Encoding	Reserved	Encoding
CodedCharSetId	Reserved	CodedCharSetId
Format	MQ format name of data that follows MQCIH	Format
Flags	Flags	Flags
ReturnCode	Return code from bridge	ReturnCode
CompCode	MQ completion code or CICS EIBRESP	CompCode
Reason	MQ reason or feedback code, or CICS EIBRESP2	Reason
UOWControl	Unit-of-work control	UOWControl
GetWaitInterval	Wait interval for MQGET call issued by bridge task	GetWaitInterval
LinkType	Link type	LinkType
OutputDataLength	Output COMMAREA data length	OutputDataLength
FacilityKeepTime	Bridge facility release time	FacilityKeepTime
ADSDescriptor	Send/receive ADS descriptor	ADSDescriptor
ConversationalTask	Whether task can be conversational	ConversationalTask
TaskEndStatus	Status at end of task	TaskEndStatus
Facility	Bridge facility token	Facility
Function	MQ call name or CICS EIBFN function	Function
AbendCode	Abend code	AbendCode
Authenticator	Password or passticket	Authenticator
Reserved1	Reserved	Reserved1
ReplyToFormat	MQ format name of reply message	ReplyToFormat
RemoteSysId	Remote CICS system Id to use	RemoteSysId
RemoteTransId	CICS RTRANSID to use	RemoteTransId
TransactionId	Transaction to attach	TransactionId
FacilityLike	Terminal emulated attributes	FacilityLike
AttentionId	AID key	AttentionId
StartCode	Transaction start code	StartCode
CancelCode	Abend transaction code	CancelCode
NextTransactionId	Next transaction to attach	NextTransactionId
Reserved2	Reserved	Reserved2
Reserved3	Reserved	Reserved3
Note: The remaining fields are not present if *Version* is less than MQCIH_VERSION_2.		
CursorPosition	Cursor position	CursorPosition
ErrorOffset	Offset of error in message	ErrorOffset

Table 130. Fields in MQCIH (continued)

Field	Description	Topic
InputItem	Reserved	InputItem
Reserved4	Reserved	Reserved4

Overview for MQCIH:

Availability: AIX, HP-UX, z/OS, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

Purpose: The MQCIH structure describes the information that can be present at the start of a message sent to the CICS bridge through WebSphere MQ for z/OS.

Format name: MQFMT_CICS.

Version: The current version of MQCIH is MQCIH_VERSION_2. Fields that exist only in the more-recent version of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQCIH, with the initial value of the *Version* field set to MQCIH_VERSION_2.

Character set and encoding: Special conditions apply to the character set and encoding used for the MQCIH structure and application message data:

- Applications that connect to the queue manager that owns the CICS bridge queue must provide an MQCIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQCIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQCIH structure that is in any of the supported character sets and encodings; the receiving message channel agent connected to the queue manager that owns the CICS bridge queue converts the MQCIH structure.
- The application message data following the MQCIH structure must be in the same character set and encoding as the MQCIH structure. You cannot use the *CodedCharSetId* and *Encoding* fields in the MQCIH structure to specify the character set and encoding of the application message data.

  You must provide a data-conversion exit to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Usage: If the application requires values that are the same as the initial values shown in Table 132 on page 1598, and the bridge is running with AUTH=LOCAL or AUTH=IDENTIFY, you can omit the MQCIH structure from the message. In all other cases, the structure must be present.

The bridge accepts either a version-1 or a version-2 MQCIH structure, but for 3270 transactions, you must use a version-2 structure.

The application must ensure that fields documented as request fields have appropriate values in the message sent to the bridge; these fields are input to the bridge.

Fields documented as response fields are set by the CICS bridge in the reply message that the bridge sends to the application. Error information is returned in the *ReturnCode*, *Function*, *CompCode*, *Reason*, and *AbendCode* fields, but not all of them are set in all cases. Table 131 on page 1588 shows which fields are set for different values of *ReturnCode*.

Table 131. Contents of error information fields in MQCIH structure for MQCIH

ReturnCode	*Function*	*CompCode*	*Reason*	*AbendCode*
MQCRC_OK	-	-	-	-
MQCRC_BRIDGE_ERROR	-	-	MQFB_CICS_*	-
MQCRC_MQ_API_ERROR MQCRC_BRIDGE_TIMEOUT	MQ call name	MQ *CompCode*	MQ *Reason*	-
MQCRC_CICS_EXEC_ERROR MQCRC_SECURITY_ERROR MQCRC_PROGRAM_NOT_AVAILABLE MQCRC_TRANSID_NOT_AVAILABLE	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	-
MQCRC_BRIDGE_ABEND MQCRC_APPLICATION_ABEND	-	-	-	CICS ABCODE

Fields for MQCIH:

The MQCIH structure contains the following fields; the fields are described in **alphabetical order**:

AbendCode (MQCHAR4):

AbendCode is a response field. The length of this field is given by MQ_ABEND_CODE_LENGTH. The initial value of this field is 4 blank characters.

The value returned in this field is significant only if the *ReturnCode* field has the value MQCRC_APPLICATION_ABEND or MQCRC_BRIDGE_ABEND. If it does, *AbendCode* contains the CICS ABCODE value.

ADSDescriptor (MQLONG):

This field is an indicator specifying whether to send ADS descriptors on SEND and RECEIVE BMS requests.

The following values are defined:

MQCADSD_NONE
Do not send or receive ADS descriptors.

MQCADSD_SEND
Send ADS descriptors.

MQCADSD_RECV
Receive ADS descriptors.

MQCADSD_MSGFORMAT
Use message format for the ADS descriptors.

This sends or receives the ADS descriptors using the long form of the ADS descriptor. The long form has fields that are aligned on 4-byte boundaries.

Set the *ADSDescriptor* field as follows:

- If you are not using ADS descriptors, set the field to MQCADSD_NONE.
- If you are using ADS descriptors with the *same* CCSID in each environment, set the field to the sum of MQCADSD_SEND and MQCADSD_RECV.
- If you are using ADS descriptors with *different* CCSIDs in each environment, set the field to the sum of MQCADSD_SEND, MQCADSD_RECV, and MQCADSD_MSGFORMAT.

This is a request field used only for 3270 transactions. The initial value of this field is MQCADSD_NONE.

AttentionId (MQCHAR4):

The value in this field determines the initial value of the AID key when the transaction is started. It is a 1 byte value, left-aligned.

AttentionId is a request field used only for 3270 transactions. The length of this field is given by MQ_ATTENTION_ID_LENGTH. The initial value of this field is four blanks.

Authenticator (MQCHAR8):

The value of this field is the password or passticket.

If user-identifier authentication is active for the CICS bridge, *Authenticator* is used with the user identifier in the MQMD identity context to authenticate the sender of the message.

This is a request field. The length of this field is given by MQ_AUTHENTICATOR_LENGTH. The initial value of this field is 8 blanks.

CancelCode (MQCHAR4):

The value in this field is the abend code to be used to terminate the transaction (normally a conversational transaction that is requesting more data). Otherwise this field is set to blanks.

This field is a request field used only for 3270 transactions. The length of this field is given by MQ_CANCEL_CODE_LENGTH. The initial value of this field is four blanks.

CodedCharSetId (MQLONG):

CodedCharSetId is a reserved field; its value is not significant. The initial value of this field is 0.

The Character Set ID for supported structures which follow an MQCIH structure is the same as the Character Set ID of the MQCIH structure itself and is taken from any preceding WebSphere MQ header.

CompCode (MQLONG):

This field is a response field. Its initial value is MQCC_OK

The value returned in this field depends on *ReturnCode*; see Table 131 on page 1588.

ConversationalTask (MQLONG):

This field is an indicator specifying whether to allow the task to issue requests for more information, or to stop the task and issue an abend message.

The value must be one of the following options:

MQCCT_YES
The task is conversational.

MQCCT_NO
The task is not conversational.

This field is a request field used only for 3270 transactions. The initial value of this field is MQCCT_NO.

CursorPosition (MQLONG):

The value in this field shows the initial cursor position when the transaction is started. For conversational transactions, the cursor position is in the RECEIVE vector.

This field is a request field used only for 3270 transactions. The initial value of this field is 0. This field is not present if *Version* is less than MQCIH_VERSION_2.

Encoding (MQLONG):

This field is a reserved field; its value is not significant. Its initial value is 0.

The Encoding for supported structures which follow an MQCIH structure is the same as the Encoding of the MQCIH structure itself and taken from any preceding WebSphere MQ header.

ErrorOffset (MQLONG):

The ErrorOffset field shows the position of invalid data detected by the bridge exit. This field provides the offset from the start of the message to the location of the invalid data.

ErrorOffset is a response field used only for 3270 transactions. The initial value of this field is 0. This field is not present if *Version* is less than MQCIH_VERSION_2.

Facility (MQBYTE8):

This field shows the 8-byte bridge facility token.

A bridge facility token enables multiple transactions in a pseudo-conversation to use the same bridge facility (virtual 3270 terminal). In the first, or only, message in a pseudo-conversation, set a value of MQCFAC_NONE. This value tells CICS to allocate a new bridge facility for this message. A bridge facility token is returned in response messages when a nonzero *FacilityKeepTime* is specified on the input message. Subsequent input messages within a pseudo-conversation must then use the same bridge facility token.

The following special value is defined:

MQCFAC_NONE
No facility token specified.

For the C programming language, the constant MQCFAC_NONE_ARRAY is also defined, and has the same value as MQCFAC_NONE, but is an array of characters instead of a string.

This field is both a request and a response field used only for 3270 transactions. The length of this field is given by MQ_FACILITY_LENGTH. The initial value of this field is MQCFAC_NONE.

FacilityKeepTime (MQLONG):

FacilityKeepTime is the length of time in seconds that the bridge facility is kept after the user transaction ends.

For pseudo-conversational transactions, specify a value that corresponds to the expected duration of a pseudo-conversation; specify zero for the last transaction of a pseudo-conversation, and for other transaction types specify zero.

This field is a request field used only for 3270 transactions. The initial value of this field is 0.

FacilityLike (MQCHAR4):

FacilityLike is the name of an installed terminal that is to be used as a model for the bridge facility.

A value of blanks means that *FacilityLike* is taken from the bridge transaction profile definition, or a default value is used.

This field is a request field used only for 3270 transactions. The length of this field is given by MQ_FACILITY_LIKE_LENGTH. The initial value of this field is four blanks.

Flags (MQLONG):

This field is a request field. The initial value of this field is MQCIH_NONE.

The value must be:

MQCIH_NONE
No flags.

MQCIH_PASS_EXPIRATION
The reply message contains:

- The same expiry report options as the request message.
- The remaining expiry time from the request message with no adjustment made for the processing time of the bridge.

If you omit this value, the expiry time is set to *unlimited*.

MQCIH_REPLY_WITHOUT_NULLS
The reply message length of a CICS DPL program request is adjusted to exclude trailing nulls (X'00') at the end of the COMMAREA returned by the DPL program. If this value is not set, the nulls might be significant, and the full COMMAREA is returned.

MQCIH_SYNC_ON_RETURN
The CICS link for DPL requests uses the SYNCONRETURN option, causing CICS to take a sync point when the program completes if it is shipped to another CICS region. The bridge does not specify to which CICS region to ship the request; that is controlled by the CICS program definition or workload balancing facilities.

Format (MQCHAR8):

This field shows the WebSphere MQ format name of the data that follows the MQCIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as the rules for coding the *Format* field in MQMD.

This format name is also used for the reply message, if the *ReplyToFormat* field has the value MQFMT_NONE.

- For DPL requests, *Format* must be the format name of the COMMAREA.
- For 3270 requests, *Format* must be `CSQCBDCI`, and the bridge sets the format to `CSQCBDCO` for Reply messages.

The data-conversion exits for these formats must be installed on the queue manager where they are to run.

If the request message generates an error reply message, the error reply message has a format name of MQFMT_STRING.

This field is a request field. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Function (MQCHAR4):

This field is a response field. The length of this field is given by MQ_FUNCTION_LENGTH. The initial value of this field is MQCFUNC_NONE.

The value returned in this field depends on *ReturnCode*; see Table 131 on page 1588. The following values are possible when *Function* contains a WebSphere MQ call name:

MQCFUNC_MQCONN
MQCONN call.

MQCFUNC_MQGET
MQGET call.

MQCFUNC_MQINQ
MQINQ call.

MQCFUNC_MQOPEN
MQOPEN call.

MQCFUNC_MQPUT
MQPUT call.

MQCFUNC_MQPUT1
MQPUT1 call.

MQCFUNC_NONE
No call.

In all cases, for the C programming language the constants MQCFUNC_*_ARRAY are also defined; these constants have the same values as the corresponding MQCFUNC_* constants, but are arrays of characters instead of strings.

GetWaitInterval (MQLONG):

This field is a request field. Its initial value is MQCGWI_DEFAULT.

This field applies only when *UOWControl* has the value MQCUOWC_FIRST. It enables the sending application to specify the approximate time in milliseconds that the MQGET calls issued by the bridge will wait for second and subsequent request messages for the unit of work started by this message. This facility overrides the default wait interval used by the bridge. You can use the following special values:

MQCGWI_DEFAULT
Default wait interval.

This value causes the CICS bridge to wait for the time specified when the bridge was started.

MQWI_UNLIMITED
Unlimited wait interval.

InputItem (MQLONG):

This field is a reserved field. The value must be 0.

This field is not present if *Version* is less than MQCIH_VERSION_2.

LinkType (MQLONG):

This field is a request field. Its initial value is MQCLT_PROGRAM.

This value indicates the type of object that the bridge tries to link. It must be one of the following values:

MQCLT_PROGRAM
DPL program.

MQCLT_TRANSACTION
3270 transaction.

NextTransactionId (MQCHAR4):

This value is the name of the next transaction returned by the user transaction (usually by EXEC CICS RETURN TRANSID). If there is no next transaction, this field is set to blanks.

This field is a response field used only for 3270 transactions. The length of this field is given by MQ_TRANSACTION_ID_LENGTH. The initial value of this field is four blanks.

OutputDataLength (MQLONG):

This field is a request field used only for DPL programs. Its initial value is MQCODL_AS_INPUT.

This value is the length of the user data to be returned to the client in a reply message. This length includes the 8-byte program name. The length of the COMMAREA passed to the linked program is the maximum of this field and the length of the user data in the request message, minus 8.

Note: The length of the user data in a message is the length of the message excluding the MQCIH structure.

If the length of the user data in the request message is smaller than *OutputDataLength*, the DATALENGTH option of the LINK command is used, enabling the LINK to be function-shipped efficiently to another CICS region.

You can use the following special value:

MQCODL_AS_INPUT
Output length is same as input length.

This value might be needed even if no reply is requested, in order to ensure that the COMMAREA passed to the linked program is of sufficient size.

Reason (MQLONG):

This field is a response field. Its initial value is MQRC_NONE.

The value returned in this field depends on *ReturnCode*; see Table 131 on page 1588.

RemoteSysId (MQCHAR4):

This field shows the CICS system identifier of the CICS system processing the request.

If this field is blank, the CICS system request is processed on the same CICS system as the bridge monitor. The SYSID used is returned in the Reply message.

For a 3270 pseudo-conversation, all subsequent messages in the conversation must specify the remote SYSID returned in the initial reply. If specified, the SYSID must:

- Be active.
- Have access to the WebSphere MQ Request queue.
- Be accessible by the CICS ISC links from the CICS system of the bridge monitor.

RemoteTransId (MQCHAR4):

This field is an optional Request field. The length of this field is given by MQ_TRANSACTION_ID_LENGTH.

If specified, the field is used as the RTRANSID value of CICS START.

ReplyToFormat (MQCHAR8):

The value of this field is the WebSphere MQ format name of the reply message that is sent in response to the current message.

The rules for coding this field are the same as those rules for coding the *Format* field in MQMD.

This field is a request field used only for DPL programs. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

Reserved1 (MQCHAR8):

This field is a reserved field. The value must be 8 blanks.

Reserved2 (MQCHAR8):

This field is a reserved field. The value must be 8 blanks.

Reserved3 (MQCHAR8):

This field is a reserved field. The value must be 8 blanks.

Reserved4 (MQLONG):

This field is a reserved field. The value must be 0.

This field is not present if *Version* is less than MQCIH_VERSION_2.

ReturnCode (MQLONG):

The value of this field is the return code from the CICS bridge describing the outcome of the processing performed by the bridge. This field is a response field, with an initial value of MQCRC_OK.

The *Function*, *CompCode*, *Reason*, and *AbendCode* fields might contain additional information (see Table 131 on page 1588). The value is one of the following:

MQCRC_APPLICATION_ABEND
(5, X'005') Application ended abnormally.

MQCRC_BRIDGE_ABEND
(4, X'004') CICS bridge ended abnormally.

MQCRC_BRIDGE_ERROR
(3, X'003') CICS bridge detected an error.

MQCRC_BRIDGE_TIMEOUT
(8, X'008') Second or later message within current unit of work not received within specified time.

MQCRC_CICS_EXEC_ERROR
(1, X'001') EXEC CICS statement detected an error.

MQCRC_MQ_API_ERROR
(2, X'002') MQ call detected an error.

MQCRC_OK
(0, X'000') No error.

MQCRC_PROGRAM_NOT_AVAILABLE
(7, X'007') Program not available.

MQCRC_SECURITY_ERROR
(6, X'006') Security error occurred.

MQCRC_TRANSID_NOT_AVAILABLE
(9, X'009') Transaction not available.

StartCode (MQCHAR4):

The value of this field is an indicator specifying whether the bridge emulates a terminal transaction or a transaction initiated with START.

The value must be one of the following:

MQCSC_START
Start.

MQCSC_STARTDATA
Start data.

MQCSC_TERMINPUT
Terminal input.

MQCSC_NONE
None.

In all cases, for the C programming language the constants MQCSC_*_ARRAY are also defined; these constants have the same values as the corresponding MQCSC_* constants, but are arrays of characters instead of strings.

In the response from the bridge, this field is set to the start code appropriate to the next transaction ID contained in the *NextTransactionId* field. The following start codes are possible in the response:

- MQCSC_START
- MQCSC_STARTDATA
- MQCSC_TERMINPUT

For CICS Transaction Server Version 1.2, this field is a request field only; its value in the response is undefined.

For CICS Transaction Server Version 1.3 and subsequent releases, this field is both a request and a response field.

This field is used only for 3270 transactions. The length of this field is given by MQ_START_CODE_LENGTH. The initial value of this field is MQCSC_NONE.

StrucId (MQCHAR4):

This field is a request field, with an initial value of MQCIH_STRUC_ID.

The value must be:

MQCIH_STRUC_ID
Identifier for CICS information header structure.

For the C programming language, the constant MQCIH_STRUC_ID_ARRAY is also defined; this has the same value as MQCIH_STRUC_ID, but is an array of characters instead of a string.

StrucLength (MQLONG):

This field is a request field, with an initial value of MQCIH_LENGTH_2.

The value must be one of the following:

MQCIH_LENGTH_1
Length of version-1 CICS information header structure.

MQCIH_LENGTH_2
Length of version-2 CICS information header structure.

The following constant specifies the length of the current version:

MQCIH_CURRENT_LENGTH
Length of current version of CICS information header structure.

TaskEndStatus (MQLONG):

This field is a response field, showing the status of the user transaction at end of task. The field is used only for 3270 transactions, and its initial value is MQCTES_NOSYNC.

One of the following values is returned:

MQCTES_NOSYNC
Not synchronized.

The user transaction has not yet completed and has not syncpointed. The *MsgType* field in MQMD is MQMT_REQUEST in this case.

MQCTES_COMMIT
Commit unit of work.

The user transaction has not yet completed, but has syncpointed the first unit of work. The *MsgType* field in MQMD is MQMT_DATAGRAM in this case.

MQCTES_BACKOUT
Back out unit of work.

The user transaction has not yet completed. The current unit of work is backed out. The *MsgType* field in MQMD is MQMT_DATAGRAM in this case.

MQCTES_ENDTASK
End task.

The user transaction has ended (or abended). The *MsgType* field in MQMD is MQMT_REPLY in this case.

TransactionId (MQCHAR4):

This field is a request field. Its length is given by MQ_TRANSACTION_ID_LENGTH. The initial value of this field is four blanks.

If *LinkType* has the value MQCLT_TRANSACTION, *TransactionId* is the transaction identifier of the user transaction to be run; specify a nonblank value in this case.

If *LinkType* has the value MQCLT_PROGRAM, *TransactionId* is the transaction code under which all programs within the unit of work are to be run. If you specify a blank value, the CICS DPL bridge default transaction code (CKBP) is used. If the value is nonblank, you must have defined it to CICS as a local transaction with an initial program that is CSQCBP00. This field applies only when *UOWControl* has the value MQCUOWC_FIRST or MQCUOWC_ONLY.

UOWControl (MQLONG):

This field is a request field which controls the unit-of-work processing performed by the CICS bridge. The initial value of this field is MQCUOWC_ONLY.

You can request the bridge to run a single transaction, or one or more programs within a unit of work. The field indicates whether the CICS bridge starts a unit of work, performs the requested function within the current unit of work, or ends the unit of work by committing it or backing it out. Various combinations are supported, to optimize the data transmission flows.

The value must be one of the following:

MQCUOWC_ONLY
Start unit of work, perform function, then commit the unit of work.

MQCUOWC_CONTINUE
Additional data for the current unit of work (3270 only).

MQCUOWC_FIRST
Start unit of work and perform function.

MQCUOWC_MIDDLE
Perform function within current unit of work

MQCUOWC_LAST
Perform function, then commit the unit of work.

MQCUOWC_COMMIT
Commit the unit of work (DPL only).

MQCUOWC_BACKOUT
Back out the unit of work (DPL only).

Version (MQLONG):

This field is a request field. Its initial value is MQCIH_VERSION_2.

The value must be one of the following:

MQCIH_VERSION_1
Version-1 CICS information header structure.

MQCIH_VERSION_2
Version-2 CICS information header structure.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCIH_CURRENT_VERSION
Current version of CICS information header structure.

Initial values and language declarations for MQCIH:

Table 132. Initial values of fields in MQCIH for MQCIH

Field name	Name of constant	Value of constant
StrucId	MQCIH_STRUC_ID	'CIH¬'
Version	MQCIH_VERSION_2	2
StrucLength	MQCIH_LENGTH_2	180
Encoding	None	0
CodedCharSetId	None	0
Format	MQFMT_NONE	Blanks
Flags	MQCIH_NONE	0
ReturnCode	MQCRC_OK	0
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
UOWControl	MQCUOWC_ONLY	273
GetWaitInterval	MQCGWI_DEFAULT	-2
LinkType	MQCLT_PROGRAM	1
OutputDataLength	MQCODL_AS_INPUT	-1
FacilityKeepTime	None	0

Table 132. Initial values of fields in MQCIH for MQCIH (continued)

Field name	Name of constant	Value of constant
ADSDescriptor	MQCADSD_NONE	0
ConversationalTask	MQCCT_NO	0
TaskEndStatus	MQCTES_NOSYNC	0
Facility	MQCFAC_NONE	Nulls
Function	MQCFUNC_NONE	Blanks
AbendCode	None	Blanks
Authenticator	None	Blanks
Reserved1	None	Blanks
ReplyToFormat	MQFMT_NONE	Blanks
RemoteSysId	None	Blanks
RemoteTransId	None	Blanks
TransactionId	None	Blanks
FacilityLike	None	Blanks
AttentionId	None	Blanks
StartCode	MQCSC_NONE	Blanks
CancelCode	None	Blanks
NextTransactionId	None	Blanks
Reserved2	None	Blanks
Reserved3	None	Blanks
CursorPosition	None	0
ErrorOffset	None	0
InputItem	None	0
Reserved4	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQCIH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQCIH MyCIH = {MQCIH_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQCIH MQCIH;
struct tagMQCIH {
  MQCHAR4  StrucId;             /* Structure identifier */
  MQLONG   Version;             /* Structure version number */
  MQLONG   StrucLength;         /* Length of MQCIH structure */
  MQLONG   Encoding;            /* Reserved */
  MQLONG   CodedCharSetId;      /* Reserved */
  MQCHAR8  Format;              /* MQ format name of data that follows
                                   MQCIH */
  MQLONG   Flags;               /* Flags */
  MQLONG   ReturnCode;          /* Return code from bridge */
  MQLONG   CompCode;            /* MQ completion code or CICS EIBRESP */
  MQLONG   Reason;              /* MQ reason or feedback code, or CICS
                                   EIBRESP2 */
  MQLONG   UOWControl;          /* Unit-of-work control */
```

```
  MQLONG   GetWaitInterval;     /* Wait interval for MQGET call issued
                                   by bridge task */
  MQLONG   LinkType;            /* Link type */
  MQLONG   OutputDataLength;    /* Output COMMAREA data length */
  MQLONG   FacilityKeepTime;    /* Bridge facility release time */
  MQLONG   ADSDescriptor;       /* Send/receive ADS descriptor */
  MQLONG   ConversationalTask;  /* Whether task can be conversational */
  MQLONG   TaskEndStatus;       /* Status at end of task */
  MQBYTE8  Facility;            /* Bridge facility token */
  MQCHAR4  Function;            /* MQ call name or CICS EIBFN
                                   function */
  MQCHAR4  AbendCode;           /* Abend code */
  MQCHAR8  Authenticator;       /* Password or passticket */
  MQCHAR8  Reserved1;           /* Reserved */
  MQCHAR8  ReplyToFormat;       /* MQ format name of reply message */
  MQCHAR4  RemoteSysId;         /* Reserved */
  MQCHAR4  RemoteTransId;       /* Reserved */
  MQCHAR4  TransactionId;       /* Transaction to attach */
  MQCHAR4  FacilityLike;        /* Terminal emulated attributes */
  MQCHAR4  AttentionId;         /* AID key */
  MQCHAR4  StartCode;           /* Transaction start code */
  MQCHAR4  CancelCode;          /* Abend transaction code */
  MQCHAR4  NextTransactionId;   /* Next transaction to attach */
  MQCHAR8  Reserved2;           /* Reserved */
  MQCHAR8  Reserved3;           /* Reserved */
  MQLONG   CursorPosition;      /* Cursor position */
  MQLONG   ErrorOffset;         /* Offset of error in message */
  MQLONG   InputItem;           /* Reserved */
  MQLONG   Reserved4;           /* Reserved */
};
```

COBOL declaration:

```
**   MQCIH structure
  10 MQCIH.
**    Structure identifier
   15 MQCIH-STRUCID            PIC X(4).
**    Structure version number
   15 MQCIH-VERSION            PIC S9(9) BINARY.
**    Length of MQCIH structure
   15 MQCIH-STRUCLENGTH        PIC S9(9) BINARY.
**    Reserved
   15 MQCIH-ENCODING           PIC S9(9) BINARY.
**    Reserved
   15 MQCIH-CODEDCHARSETID     PIC S9(9) BINARY.
**    MQ format name of data that follows MQCIH
   15 MQCIH-FORMAT             PIC X(8).
**    Flags
   15 MQCIH-FLAGS              PIC S9(9) BINARY.
**    Return code from bridge
   15 MQCIH-RETURNCODE         PIC S9(9) BINARY.
**    MQ completion code or CICS EIBRESP
   15 MQCIH-COMPCODE           PIC S9(9) BINARY.
**    MQ reason or feedback code, or CICS EIBRESP2
   15 MQCIH-REASON             PIC S9(9) BINARY.
**    Unit-of-work control
   15 MQCIH-UOWCONTROL         PIC S9(9) BINARY.
**    Wait interval for MQGET call issued by bridge task
   15 MQCIH-GETWAITINTERVAL    PIC S9(9) BINARY.
**    Link type
   15 MQCIH-LINKTYPE           PIC S9(9) BINARY.
**    Output COMMAREA data length
   15 MQCIH-OUTPUTDATALENGTH   PIC S9(9) BINARY.
**    Bridge facility release time
   15 MQCIH-FACILITYKEEPTIME   PIC S9(9) BINARY.
**    Send/receive ADS descriptor
   15 MQCIH-ADSDESCRIPTOR      PIC S9(9) BINARY.
```

```
**    Whether task can be conversational
   15 MQCIH-CONVERSATIONALTASK PIC S9(9) BINARY.
**    Status at end of task
   15 MQCIH-TASKENDSTATUS      PIC S9(9) BINARY.
**    Bridge facility token
   15 MQCIH-FACILITY           PIC X(8).
**    MQ call name or CICS EIBFN function
   15 MQCIH-FUNCTION           PIC X(4).
**    Abend code
   15 MQCIH-ABENDCODE          PIC X(4).
**    Password or passticket
   15 MQCIH-AUTHENTICATOR      PIC X(8).
**    Reserved
   15 MQCIH-RESERVED1          PIC X(8).
**    MQ format name of reply message
   15 MQCIH-REPLYTOFORMAT      PIC X(8).
**    Reserved
   15 MQCIH-REMOTESYSID        PIC X(4).
**    Reserved
   15 MQCIH-REMOTETRANSID      PIC X(4).
**    Transaction to attach
   15 MQCIH-TRANSACTIONID      PIC X(4).
**    Terminal emulated attributes
   15 MQCIH-FACILITYLIKE       PIC X(4).
**    AID key
   15 MQCIH-ATTENTIONID        PIC X(4).
**    Transaction start code
   15 MQCIH-STARTCODE          PIC X(4).
**    Abend transaction code
   15 MQCIH-CANCELCODE         PIC X(4).
**    Next transaction to attach
   15 MQCIH-NEXTTRANSACTIONID  PIC X(4).
**    Reserved
   15 MQCIH-RESERVED2          PIC X(8).
**    Reserved
   15 MQCIH-RESERVED3          PIC X(8).
**    Cursor position
   15 MQCIH-CURSORPOSITION     PIC S9(9) BINARY.
**    Offset of error in message
   15 MQCIH-ERROROFFSET        PIC S9(9) BINARY.
**    Reserved
   15 MQCIH-INPUTITEM          PIC S9(9) BINARY.
**    Reserved
   15 MQCIH-RESERVED4          PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQCIH based,
  3 StrucId            char(4),       /* Structure identifier */
  3 Version            fixed bin(31), /* Structure version number */
  3 StrucLength        fixed bin(31), /* Length of MQCIH structure */
  3 Encoding           fixed bin(31), /* Reserved */
  3 CodedCharSetId     fixed bin(31), /* Reserved */
  3 Format             char(8),       /* MQ format name of data that
                                         follows MQCIH */
  3 Flags              fixed bin(31), /* Flags */
  3 ReturnCode         fixed bin(31), /* Return code from bridge */
  3 CompCode           fixed bin(31), /* MQ completion code or CICS
                                         EIBRESP */
  3 Reason             fixed bin(31), /* MQ reason or feedback code, or
                                         CICS EIBRESP2 */
  3 UOWControl         fixed bin(31), /* Unit-of-work control */
  3 GetWaitInterval    fixed bin(31), /* Wait interval for MQGET call
                                         issued by bridge task */
  3 LinkType           fixed bin(31), /* Link type */
  3 OutputDataLength   fixed bin(31), /* Output COMMAREA data length */
```

```
  3 FacilityKeepTime   fixed bin(31), /* Bridge facility release time */
  3 ADSDescriptor      fixed bin(31), /* Send/receive ADS descriptor */
  3 ConversationalTask fixed bin(31), /* Whether task can be
                                         conversational */
  3 TaskEndStatus      fixed bin(31), /* Status at end of task */
  3 Facility           char(8),       /* Bridge facility token */
  3 Function           char(4),       /* MQ call name or CICS EIBFN
                                         function */
  3 AbendCode          char(4),       /* Abend code */
  3 Authenticator      char(8),       /* Password or passticket */
  3 Reserved1          char(8),       /* Reserved */
  3 ReplyToFormat      char(8),       /* MQ format name of reply
                                         message */
  3 RemoteSysId        char(4),       /* Reserved */
  3 RemoteTransId      char(4),       /* Reserved */
  3 TransactionId      char(4),       /* Transaction to attach */
  3 FacilityLike       char(4),       /* Terminal emulated attributes */
  3 AttentionId        char(4),       /* AID key */
  3 StartCode          char(4),       /* Transaction start code */
  3 CancelCode         char(4),       /* Abend transaction code */
  3 NextTransactionId  char(4),       /* Next transaction to attach */
  3 Reserved2          char(8),       /* Reserved */
  3 Reserved3          char(8),       /* Reserved */
  3 CursorPosition     fixed bin(31), /* Cursor position */
  3 ErrorOffset        fixed bin(31), /* Offset of error in message */
  3 InputItem          fixed bin(31), /* Reserved */
  3 Reserved4          fixed bin(31); /* Reserved */
```

High Level Assembler declaration:

```
MQCIH                   DSECT
MQCIH_STRUCID           DS   CL4  Structure identifier
MQCIH_VERSION           DS   F    Structure version number
MQCIH_STRUCLENGTH       DS   F    Length of MQCIH structure
MQCIH_ENCODING          DS   F    Reserved
MQCIH_CODEDCHARSETID    DS   F    Reserved
MQCIH_FORMAT            DS   CL8  MQ format name of data that follows
*                                 MQCIH
MQCIH_FLAGS             DS   F    Flags
MQCIH_RETURNCODE        DS   F    Return code from bridge
MQCIH_COMPCODE          DS   F    MQ completion code or CICS EIBRESP
MQCIH_REASON            DS   F    MQ reason or feedback code, or CICS
*                                 EIBRESP2
MQCIH_UOWCONTROL        DS   F    Unit-of-work control
MQCIH_GETWAITINTERVAL   DS   F    Wait interval for MQGET call issued
*                                 by bridge task
MQCIH_LINKTYPE          DS   F    Link type
MQCIH_OUTPUTDATALENGTH  DS   F    Output COMMAREA data length
MQCIH_FACILITYKEEPTIME  DS   F    Bridge facility release time
MQCIH_ADSDESCRIPTOR     DS   F    Send/receive ADS descriptor
MQCIH_CONVERSATIONALTASK DS  F    Whether task can be conversational
MQCIH_TASKENDSTATUS     DS   F    Status at end of task
MQCIH_FACILITY          DS   XL8  Bridge facility token
MQCIH_FUNCTION          DS   CL4  MQ call name or CICS EIBFN function
MQCIH_ABENDCODE         DS   CL4  Abend code
MQCIH_AUTHENTICATOR     DS   CL8  Password or passticket
MQCIH_RESERVED1         DS   CL8  Reserved
MQCIH_REPLYTOFORMAT     DS   CL8  MQ format name of reply message
MQCIH_REMOTESYSID       DS   CL4  Reserved
MQCIH_REMOTETRANSID     DS   CL4  Reserved
MQCIH_TRANSACTIONID     DS   CL4  Transaction to attach
MQCIH_FACILITYLIKE      DS   CL4  Terminal emulated attributes
MQCIH_ATTENTIONID       DS   CL4  AID key
MQCIH_STARTCODE         DS   CL4  Transaction start code
MQCIH_CANCELCODE        DS   CL4  Abend transaction code
MQCIH_NEXTTRANSACTIONID DS   CL4  Next transaction to attach
MQCIH_RESERVED2         DS   CL8  Reserved
```

```
MQCIH_RESERVED3           DS   CL8  Reserved
MQCIH_CURSORPOSITION      DS   F    Cursor position
MQCIH_ERROROFFSET         DS   F    Offset of error in message
MQCIH_INPUTITEM           DS   F    Reserved
MQCIH_RESERVED4           DS   F    Reserved
*
MQCIH_LENGTH              EQU  *-MQCIH
                          ORG  MQCIH
MQCIH_AREA                DS   CL(MQCIH_LENGTH)
```

Visual Basic declaration:

```
Type MQCIH
  StrucId            As String*4 'Structure identifier'
  Version            As Long     'Structure version number'
  StrucLength        As Long     'Length of MQCIH structure'
  Encoding           As Long     'Reserved'
  CodedCharSetId     As Long     'Reserved'
  Format             As String*8 'MQ format name of data that follows'
                                 'MQCIH'
  Flags              As Long     'Flags'
  ReturnCode         As Long     'Return code from bridge'
  CompCode           As Long     'MQ completion code or CICS EIBRESP'
  Reason             As Long     'MQ reason or feedback code, or CICS'
                                 'EIBRESP2'
  UOWControl         As Long     'Unit-of-work control'
  GetWaitInterval    As Long     'Wait interval for MQGET call issued'
                                 'by bridge task'
  LinkType           As Long     'Link type'
  OutputDataLength   As Long     'Output COMMAREA data length'
  FacilityKeepTime   As Long     'Bridge facility release time'
  ADSDescriptor      As Long     'Send/receive ADS descriptor'
  ConversationalTask As Long     'Whether task can be conversational'
  TaskEndStatus      As Long     'Status at end of task'
  Facility           As MQBYTE8  'Bridge facility token'
  Function           As String*4 'MQ call name or CICS EIBFN function'
  AbendCode          As String*4 'Abend code'
  Authenticator      As String*8 'Password or passticket'
  Reserved1          As String*8 'Reserved'
  ReplyToFormat      As String*8 'MQ format name of reply message'
  RemoteSysId        As String*4 'Reserved'
  RemoteTransId      As String*4 'Reserved'
  TransactionId      As String*4 'Transaction to attach'
  FacilityLike       As String*4 'Terminal emulated attributes'
  AttentionId        As String*4 'AID key'
  StartCode          As String*4 'Transaction start code'
  CancelCode         As String*4 'Abend transaction code'
  NextTransactionId  As String*4 'Next transaction to attach'
  Reserved2          As String*8 'Reserved'
  Reserved3          As String*8 'Reserved'
  CursorPosition     As Long     'Cursor position'
  ErrorOffset        As Long     'Offset of error in message'
  InputItem          As Long     'Reserved'
  Reserved4          As Long     'Reserved'
End Type
```

MQCMHO - Create message handle options:

The following table summarizes the fields in the structure.

Table 133. Fields in MQCMHO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options

Overview for MQCMHO:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS and WebSphere MQ clients.

Purpose: The **MQCMHO** structure allows applications to specify options that control how message handles are created. The structure is an input parameter on the **MQCRTMH** call.

Character set and encoding: Data in **MQCMHO** must be in the character set of the application and encoding of the application (**MQENC_NATIVE**).

Fields for MQCMHO:

The MQCMHO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

This field is always an input field. Its initial value is MQCMHO_DEFAULT_VALIDATION.

One of the following options can be specified:

MQCMHO_VALIDATE

When **MQSETMP** is called to set a property in this message handle, the property name is validated to ensure that it:

- contains no invalid characters.
- does not begin JMS or usr.JMS except for the following:
  - JMSCorrelationID
  - JMSReplyTo
  - JMSType
  - JMSXGroupID
  - JMSXGroupSeq

  These names are reserved for JMS properties.
- is not one of the following keywords, in any mixture of upper or lowercase:
  - AND
  - BETWEEN
  - ESCAPE
  - FALSE
  - IN
  - IS
  - LIKE
  - NOT

- NULL
- OR
- TRUE

- does not begin Body. or Root. (except for Root.MQMD.).

If the property is MQ-defined (mq.*) and the name is recognized, the property descriptor fields are set to the correct values for the property. If the property is not recognized, the *Support* field of the property descriptor is set to **MQPD_OPTIONAL**.

MQCMHO_DEFAULT_VALIDATION

This value specifies that the default level of validation of property names occur.

The default level of validation is equivalent to the level specified by **MQCMHO_VALIDATE**.

This value is the default value.

MQCMHO_NO_VALIDATION

No validation on the property name occurs. See the description of **MQCMHO_VALIDATE**.

Default option: If none of the preceding options described is required, the following option can be used:

MQCMHO_NONE

All options assume their default values. Use this value to indicate that no other options have been specified. **MQCMHO_NONE** aids program documentation; it is not intended that this option is used with any other, but as its value is zero, such use cannot be detected.

StrucId (MQCHAR4):

This field is always an input field. Its initial value is MQCMHO_STRUC_ID.

This is the structure identifier; the value must be:

MQCMHO_STRUC_ID
Identifier for create message handle options structure.

For the C programming language, the constant **MQCMHO_STRUC_ID_ARRAY** is also defined; this has the same value as **MQCMHO_STRUC_ID**, but is an array of characters instead of a string.

Version (MQLONG):

This field is always an input field. Its initial value is MQCMHO_VERSION_1.

This is the structure version number; the value must be:

MQCMHO_VERSION_1
Version-1 create message handle options structure.

The following constant specifies the version number of the current version:

MQCMHO_CURRENT_VERSION
Current version of create message handle options structure.

Initial values and language declarations for MQCMHO:

Table 134. Initial values of fields in MQCMHO

Field name	Name of constant	Value of constant
StrucId	MQCMHO_STRUC_ID	'CMHO'
Version	MQCMHO_VERSION_1	1
Options	MQCMHO_DEFAULT_VALIDATION	0

Notes:

1. In the C programming language, the macro variable MQCMHO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

   ```
   MQCMHO MyCMHO = {MQCMHO_DEFAULT};
   ```

C declaration:

```
struct tagMQCMHO {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Options;         /* Options that control the action of MQCRTMH */
};
```

COBOL declaration:

```
**   MQCMHO structure
  10 MQCMHO.
**    Structure identifier
   15 MQCMHO-STRUCID        PIC X(4).
**    Structure version number
   15 MQCMHO-VERSION        PIC S9(9) BINARY.
**    Options that control the action of MQCRTMH
   15 MQCMHO-OPTIONS        PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQCMHO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of MQCRTMH */
```

High Level Assembler declaration:

```
MQCMHO                DSECT
MQCMHO_STRUCID        DS   CL4   Structure identifier
MQCMHO_VERSION        DS   F     Structure version number
MQCMHO_OPTIONS        DS   F     Options that control the action of
*                                MQCRTMH
MQCMHO_LENGTH         EQU  *-MQCMHO
MQCMHO_AREA           DS   CL(MQCMHO_LENGTH)
```

MQCNO - Connect options:

The following table summarizes the fields in the structure.

Table 135. Fields in MQCNO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options that control the action of MQCONNX	Options
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_2.		
ClientConnOffset	Offset of MQCD structure for client connection	ClientConnOffset
ClientConnPtr	Address of MQCD structure for client connection	ClientConnPtr
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_3.		
ConnTag	Queue-manager connection tag	ConnTag
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_4.		
SSLConfigPtr	Address of MQSCO structure for client connection	SSLConfigPtr
SSLConfigOffset	Offset of MQSCO structure for client connection	SSLConfigOffset
Note: The remaining fields are ignored if *Version* is less than MQCNO_VERSION_5.		
ConnectionId	Unique connection ID	ConnectionId
SecurityParmsOffset	Security parameters	SecurityParmsOffset
SecurityParmsPtr	Security parameters	SecurityParmsPtr

Related information:
Using MQCONNX

Overview for MQCNO:

Availability: All versions except MQCNO_VERSION_4: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

Purpose: The MQCNO structure allows the application to specify options relating to the connection to the local queue manager. The structure is an input/output parameter on the MQCONNX call. For more information about using shared handles, and the MQCONNX call, see Shared (thread independent) connections with MQCONNX.

Version: The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQCNO, but with the initial value of the *Version* field set to MQCNO_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

Character set and encoding: Data in MQCNO must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as a WebSphere MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQCNO:

The MQCNO structure contains the following fields; the fields are described in **alphabetical order**:

ClientConnOffset (MQLONG):

ClientConnOffset is the offset in bytes of an MQCD channel definition structure from the start of the MQCNO structure. The offset can be positive or negative. This field is an input field with an initial value of 0.

Use *ClientConnOffset* only when the application issuing the MQCONNX call is running as a WebSphere MQ MQI client. For information about how to use this field, see the description of the *ClientConnPtr* field.

This field is ignored if *Version* is less than MQCNO_VERSION_2.

ClientConnPtr (MQPTR):

ClientConnPtr is an input field. Its initial value is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Use *ClientConnOffset* and *ClientConnPtr* only when the application issuing the MQCONNX call is running as a WebSphere MQ MQI client. By specifying one or other of these fields, the application can control the definition of the client connection channel by providing an MQCD channel definition structure that contains the values required.

If the application is running as a WebSphere MQ MQI client, but does not provide an MQCD structure, the MQSERVER environment variable is used to select the channel definition. If MQSERVER is not set, the client channel table is used.

If the application is not running as a WebSphere MQ MQI client, *ClientConnOffset* and *ClientConnPtr* are ignored.

If the application provides an MQCD structure, set the fields listed to the values required; other fields in MQCD are ignored. You can pad character strings with blanks to the length of the field, or terminated them with a null character. See “Fields” on page 2349 for more information about the fields in the MQCD structure.

Field in MQCD	Value
ChannelName	Channel name.
Version	Structure version number. Must not be less than MQCD_VERSION_7.
TransportType	Any supported transport type.
ModeName	LU 6.2 mode name.
TpName	LU 6.2 transaction program name.
SecurityExit	Name of channel security exit.
SendExit	Name of channel send exit.
ReceiveExit	Name of channel receive exit.
MaxMsgLength	Maximum length in bytes of messages that can be sent over the client connection channel.
SecurityUserData	User data for security exit.
SendUserData	User data for send exit.
ReceiveUserData	User data for receive exit.
UserIdentifier	User identifier to be used to establish an LU 6.2 session.
Password	Password to be used to establish an LU 6.2 session.
ConnectionName	Connection name.
HeartbeatInterval	Time in seconds between heartbeat flows.

Field in MQCD	Value
StrucLength	Length of the MQCD structure.
ExitNameLength	Length of exit names addressed by *SendExitPtr* and *ReceiveExitPtr*. Must be greater than zero if *SendExitPtr* or *ReceiveExitPtr* is set to a value that is not the null pointer.
ExitDataLength	Length of exit data addressed by *SendUserDataPtr* and *ReceiveUserDataPtr*. Must be greater than zero if *SendUserDataPtr* or *ReceiveUserDataPtr* is set to a value that is not the null pointer.
SendExitsDefined	Number of send exits addressed by *SendExitPtr*. If zero, *SendExit* and *SendUserData* provide the exit name and data. If greater than zero, *SendExitPtr* and *SendUserDataPtr* provide the exit names and data, and *SendExit* and *SendUserData* must be blank.
ReceiveExitsDefined	Number of receive exits addressed by *ReceiveExitPtr*. If zero, *ReceiveExit* and *ReceiveUserData* provide the exit name and data. If greater than zero, *ReceiveExitPtr* and *ReceiveUserDataPtr* provide the exit names and data, and *ReceiveExit* and *ReceiveUserData* must be blank.
SendExitPtr	Address of name of first send exit.
SendUserDataPtr	Address of data for first send exit.
ReceiveExitPtr	Address of name of first receive exit.
ReceiveUserDataPtr	Address of data for first receive exit.
LongRemoteUserIdLength	Length of long remote user identifier.
LongRemoteUserIdPtr	Address of long remote user identifier.
RemoteSecurityId	Remote security identifier.
SSLCipherSpec	SSL CipherSpec.
SSLPeerNamePtr	Address of SSL peer name.
SSLPeerNameLength	Length of SSL peer name.
KeepAliveInterval	Value passed to the communications stack for keepalive timing for the channel
LocalAddress	The local communications address, including the IP address of the local network adapter to use, and a range of ports to use for outgoing connections.

Provide the channel definition structure in one of two ways:

- By using the offset field *ClientConnOffset*

  In this case, the application must declare a compound structure containing an MQCNO followed by the channel definition structure MQCD, and set *ClientConnOffset* to the offset of the channel definition structure from the start of the MQCNO. Ensure that this offset is correct. *ClientConnPtr* must be set to the null pointer or null bytes.

  Use *ClientConnOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a way that is not portable to different environments (for example, the COBOL programming language).

  For the Visual Basic programming language, a compound structure called MQCNOCD is provided in the header file CMQXB.BAS; this structure contains an MQCNO structure followed by an MQCD structure. Initialize MQCNOCD by invoking the MQCNOCD_DEFAULTS subroutine. MQCNOCD is used with the MQCONNXAny variant of the MQCONNX call; see the description of the MQCONNX call for further details.

- By using the pointer field *ClientConnPtr*

  In this case, the application can declare the channel definition structure separately from the MQCNO structure, and set *ClientConnPtr* to the address of the channel definition structure. Set *ClientConnOffset* to zero.

  Use *ClientConnPtr* for programming languages that support the pointer data type in a way that is portable to different environments (for example, the C programming language).

  In the C programming language, you can use the macro variable MQCD_CLIENT_CONN_DEFAULT to provide initial values for the structure that are more suitable for use on the MQCONNX call than the initial values provided by MQCD_DEFAULT.

Whichever technique you choose, you can use only one of *ClientConnOffset* and *ClientConnPtr*; the call fails with reason code MQRC_CLIENT_CONN_ERROR if both are nonzero.

When the MQCONNX call has completed, the MQCD structure is not referenced again.

This field is ignored if *Version* is less than MQCNO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

ConnectionId (MQBYTE24):

ConnectionId is a unique 24-byte identifier that allows WebSphere MQ to reliably identify an application. An application can use this identifier for correlation in PUT and GET calls. This output parameter has an initial value of 24 null bytes in all programming languages.

The queue manager assigns a unique ID to all connections, however they are established. If an MQCONNX establishes the connection with a version 5 MQCNO, the application can determine the ConnectionId from the returned MQCNO. The assigned identifier is guaranteed to be unique among all other identifiers that WebSphere MQ generates, such as CorrelId, MsgID, and GroupId.

Use the ConnectionId to identify long running units of work using the PCF command Inquire Connection or the MQSC command DISPLAY CONN. The ConnectionId used by MQSC commands (CONN) is derived from the ConnectionId returned here. The PCF Inquire and Stop Connection commands can use the ConnectionId returned here without modification.

You can use the ConnectionId to force the end of a long running unit of work, by specifying the ConnectionId using the PCF command Stop Connection or the MQSC command STOP CONN. See Stop Connection and STOP CONN for more information about using these commands.

This field is not returned if Version is less than MQCNO_VERSION_5.

The length of this field is given by MQ_CONNECTION_ID_LENGTH.

ConnTag (MQBYTE128):

ConnTag is a tag that the queue manager associates with the resources that are affected by the application during this connection. Each application or application instance must use a different value for the tag, so that the queue manager can correctly serialize access to the affected resources. This field is an input field, and its initial value is MQCT_NONE.

See the descriptions of the MQCNO_*_CONN_TAG_* options for further details about values to be used by different applications. The tag ceases to be valid when the application terminates or issues the MQDISC call.

Note: Connection tag values beginning with `MQ` in upper, lower, or mixed case in either ASCII or EBCDIC are reserved for use by IBM products. Do not use connection tag values beginning with these letters.

Use the following special value if you require no tag:

MQCT_NONE
The value is binary zero for the length of the field.

For the C programming language, the constant MQCT_NONE_ARRAY is also defined; this constant has the same value as MQCT_NONE, but is an array of characters instead of a string.

This field is used when connecting to a z/OS queue manager. In other environments, specify the value MQCT_NONE.

The length of this field is given by MQ_CONN_TAG_LENGTH. This field is ignored if *Version* is less than MQCNO_VERSION_3.

Options (MQLONG):

Options that control the action of MQCONNX.

Accounting options

The following options control the type of accounting if the *AccountingConnOverride* queue manager attribute is set to MQMON_ENABLED:

MQCNO_ACCOUNTING_MQI_ENABLED
When monitoring data collection is switched off in the queue manager definition by setting the *MQIAccounting* attribute to MQMON_OFF, setting this flag enables MQI accounting data collection.

MQCNO_ACCOUNTING_MQI_DISABLED
When monitoring data collection is switched off in the queue manager definition by setting the *MQIAccounting* attribute to MQMON_OFF, setting this flag stops MQI accounting data collection.

MQCNO_ACCOUNTING_Q_ENABLED
When queue-accounting data collection is switched off in the queue manager definition by setting the *MQIAccounting* attribute to MQMON_OFF, setting this flag enables accounting data collection for those queues that specify a queue manager in the *MQIAccounting* field of their queue definition.

MQCNO_ACCOUNTING_Q_DISABLED
When queue-accounting data collection is switched off in the queue manager definition by setting the *MQIAccounting* attribute to MQMON_OFF, setting this flag switches off accounting data collection for those queues that specify a queue manager in the *MQIAccounting* field of their queue definition.

If none of these flags are defined, the accounting for the connection is as defined in the Queue Manager attributes.

Binding options

The following options control the type of WebSphere MQ binding to use. Specify only one of these options:

MQCNO_STANDARD_BINDING
The application and the local queue manager agent (the component that manages queuing operations) run in separate units of execution (typically, in separate processes). This arrangement maintains the integrity of the queue manager; that is, it protects the queue manager from errant programs.

If the queue manager supports multiple binding types, and you set MQCNO_STANDARD_BINDING, the queue manager uses the *DefaultBindType* attribute in the *Connection* stanza in the `qm.ini` file (or the equivalent Windows registry entry) to select the actual type of binding. If this stanza is not defined, or the value cannot be used or is not appropriate for the application, the queue manager selects an appropriate binding type. The queue manager sets the actual binding type used in the connect options.

Use MQCNO_STANDARD_BINDING in situations where the application might not have been fully tested, or might be unreliable or untrustworthy. MQCNO_STANDARD_BINDING is the default.

This option is supported in all environments.

If you are linking to the mqm library, then a standard server connection using the default bind type is attempted first. If the underlying server library failed to load, a client connection is attempted instead.

- If the MQ_CONNECT_TYPE environment variable is specified, one of the following options can be supplied to change the behaviour of MQCONN or MQCONNX if MQCNO_STANDARD_BINDING is specified. (The exception to this is if MQCNO_FASTPATH_BINDING is specified with MQ_CONNECT_TYPE set to `LOCAL` or `STANDARD` to allow fastpath connections to be downgraded by the administrator without a related change to the application:

Value	Meaning
CLIENT	A client connection only is attempted.
FASTPATH	This value was supported in previous releases, but is now ignored if specified.
LOCAL	A server connection only is attempted. Fastpath connections are downgraded to a standard server connection.
STANDARD	Supported for compatibility with previous releases. This value is now treated as LOCAL.

- If the MQ_CONNECT_TYPE environment variable is not set when MQCONNX is called, a standard server connection using the default bind type is attempted. If the server library fails to load, a client connection is attempted.

MQCNO_FASTPATH_BINDING

The application and the local queue manager agent are part of the same unit of execution. This is in contrast to the typical method of binding, where the application and the local queue manager agent run in separate units of execution.

MQCNO_FASTPATH_BINDING is ignored if the queue manager does not support this type of binding; processing continues as though the option had not been specified.

MQCNO_FASTPATH_BINDING can be of advantage in situations where multiple processes consume more resources than the overall resource used by the application. An application that uses the fastpath binding is known as a *trusted application*.

Consider the following important points when deciding whether to use the fastpath binding:

- Using the MQCNO_FASTPATH_BINDING option does not prevent an application altering or corrupting messages and other data areas belonging to the queue manager. Use this option only in situations where you have fully evaluated these issues.
- The application must not use asynchronous signals or timer interrupts (such as `sigkill`) with MQCNO_FASTPATH_BINDING. There are also restrictions on the use of shared memory segments.
- The application must use the MQDISC call to disconnect from the queue manager.
- The application must finish before you end the queue manager with the `endmqm` command.
- On IBM i, the job must run under a user profile that belongs to the `QMQMADM` group. Also, the program must not stop abnormally, otherwise unpredictable results can occur.
- On UNIX systems, the `mqm` user identifier must be the effective user identifier, and the `mqm` group identifier must be the effective group identifier. To make the application run in this way, configure the program so that it is owned by the `mqm` user identifier and `mqm` group identifier, and then set the `setuid` and `setgid` permission bits on the program.

  The WebSphere MQ Object Authority Manager (OAM) still uses the real user ID for authority checking.

- On Windows, the program must be a member of the mqm group. Fastpath binding is not supported for 64 bit applications.

The MQCNO_FASTPATH_BINDING option is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. On z/OS, the option is accepted but ignored.

For more information about the implications of using trusted applications, see Restrictions for trusted applications.

MQCNO_SHARED_BINDING

With MQCNO_SHARED_BINDING, the application and the local-queue-manager agent share some resources. MQCNO_SHARED_BINDING is ignored if the queue manager does not support this type of binding. Processing continues as though the option had not been specified.

MQCNO_ISOLATED_BINDING

In this case, the application process and the local queue manager agent are isolated from each other in that they do not share resources. MQCNO_ISOLATED_BINDING is ignored if the queue manager does not support this type of binding. Processing continues as though the option had not been specified.

MQCNO_CLIENT_BINDING

Specify this option to make the application attempt a client connection only. This option has the following limitations:

- MQCNO_CLIENT_BINDING is rejected on z/OS with MQRC_OPTIONS_ERROR.
- MQCNO_CLIENT_BINDING is rejected with MQRC_OPTIONS_ERROR if it is specified with any MQCNO binding option other than MQCNO_STANDARD_BINDING.
- MQCNO_CLIENT_BINDING is not available for Java or .NET as they have their own mechanisms for choosing the bind type.
- If the MQ_CONNECT_TYPE environment variable is not set when MQCONNX is called, a standard server connection using the default bind type is attempted. If the server library fails to load, a client connection is attempted.

MQCNO_LOCAL_BINDING

Specify this option to make the application attempt a server connection. If either MQCNO_FASTPATH_BINDING, MQCNO_ISOLATED_BINDING, or MQCNO_SHARED_BINDING is also specified, then the connection is of that type instead, and is documented in this section. Otherwise a standard server connection is attempted using the default bind type. MQCNO_LOCAL_BINDING has the following limitations:

- MQCNO_LOCAL_BINDING is ignored on z/OS.
- MQCNO_LOCAL_BINDING is rejected with MQRC_OPTIONS_ERROR if it is specified with any MQCNO reconnect option other than MQCNO_RECONNECT_AS_DEF.
- MQCNO_LOCAL_BINDING is not available for Java or .NET as they have their own mechanisms for choosing the bind type.
- If the MQ_CONNECT_TYPE environment variable is not set when MQCONNX is called, a standard server connection using the default bind type is attempted. If the server library fails to load, a client connection is attempted.

On AIX, HP-UX, Solaris, Linux, and Windows, you can use the environment variable MQ_CONNECT_TYPE with the bind type specified by the *Options* field, to control the type of binding used. If you specify this environment variable, it must have the value FASTPATH or STANDARD ; if it has a different value, it is ignored. The value of the environment variable is case sensitive; see MQCONNX environment variable for more information.

The environment variable and *Options* field interact as follows:

- If you omit the environment variable, or give it a value that is not supported, use of the fastpath binding is determined solely by the *Options* field.
- If you give the environment variable a supported value, the fastpath binding is used only if *both* the environment variable and *Options* field specify the fastpath binding.

Connection-tag options

These options are supported only when connecting to a z/OS queue manager and they control the use of the connection tag *ConnTag*. You can specify only one of these options:

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

This option requests exclusive use of the connection tag within the local queue manager. If the connection tag is already in use in the local queue manager, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE. The outcome of the call is not affected by using the connection tag elsewhere in the queue-sharing group to which the local queue manager belongs.

MQCNO_SERIALIZE_CONN_TAG_QSG

This option requests exclusive use of the connection tag within the queue-sharing group to which the local queue manager belongs. If the connection tag is already in use in the queue-sharing group, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

This option requests shared use of the connection tag within the local queue manager. If the connection tag is already in use in the local queue manager, the MQCONNX call can succeed if the requesting application is running in the same processing scope as the existing user of the tag. If this condition is not satisfied, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE. The outcome of the call is not affected by use of the connection tag elsewhere in the queue-sharing group to which the local queue manager belongs.

- Applications must run within the same MVS address space to share the connection tag. If the application using the connection tag is a client application, MQCNO_RESTRICT_CONN_TAG_Q_MGR is not allowed.

MQCNO_RESTRICT_CONN_TAG_QSG

This option requests shared use of the connection tag within the queue-sharing group to which the local queue manager belongs. If the connection tag is already in use in the queue-sharing group, the MQCONNX call can succeed provided the requesting application is running in the same processing scope and is connected to the same queue manager, as the existing user of the tag.

If these conditions are not satisfied, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE.

- Applications must run within the same MVS address space to share the connection tag. If the application using the connection tag is a client application, MQCNO_RESTRICT_CONN_TAG_QSG is not allowed.

If none of these options are specified, *ConnTag* is not used. These options are not valid if *Version* is less than MQCNO_VERSION_3.

Handle-sharing options

These options are supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, and Windows. They control the sharing of handles between different threads (units of parallel processing) within the same process. You can specify only one of these options:

MQCNO_HANDLE_SHARE_NONE

This option indicates that connection and object handles can be used only by the thread that caused the handle to be allocated (that is, the thread that issued the MQCONN, MQCONNX, or MQOPEN call). The handles cannot be used by other threads belonging to the same process.

MQCNO_HANDLE_SHARE_BLOCK

This option indicates that connection and object handles allocated by one thread of a process can be used by other threads belonging to the same process. However, only one thread at a time can use any particular handle; that is, only serial use of a handle is permitted. If a thread tries to use a handle that is already in use by another thread, the call blocks (waits) until the handle becomes available.

MQCNO_HANDLE_SHARE_NO_BLOCK

This is the same as MQCNO_HANDLE_SHARE_BLOCK, except that if the handle is in use by another thread, the call completes immediately with MQCC_FAILED and MQRC_CALL_IN_PROGRESS instead of blocking until the handle becomes available.

A thread can have zero or one non-shared handles:

- Each MQCONN or MQCONNX call that specifies MQCNO_HANDLE_SHARE_NONE returns a new nonshared handle on the first call, and the same non-shared handle on the second and later calls (assuming no intervening MQDISC call). The reason code is MQRC_ALREADY_CONNECTED for the second and later calls.
- Each MQCONNX call that specifies MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK returns a new shared handle on each call.

Object handles inherit the same sharing properties as the connection handle specified on the MQOPEN call that created the object handle. Also, units of work inherit the same sharing properties as the connection handle used to start the unit of work; if the unit of work is started in one thread using a shared handle, the unit of work can be updated in another thread using the same handle.

If you do not specify a handle-sharing option, the default is determined by the environment:

- In the Microsoft Transaction Server (MTS) environment, the default is the same as MQCNO_HANDLE_SHARE_BLOCK.
- In other environments, the default is the same as MQCNO_HANDLE_SHARE_NONE.

Reconnection options

Reconnection options determine if a connection is reconnectable. Only client connections are reconnectable.

MQCNO_RECONNECT_AS_DEF

The reconnection option is resolved to its default value. If no default is set, the value of this option resolves to DISABLED . The value of the option is passed to the server, and can be queried by PCF and MQSC.

MQCNO_RECONNECT

The application can be reconnected to any queue manager consistent with the value of the `QmgrName` parameter of MQCONNX. Use the MQCNO_RECONNECT option only if there is no affinity between the client application and the queue manager with which it initially established a connection. The value of the option is passed to the server, and can be queried by PCF and MQSC.

MQCNO_RECONNECT_DISABLED

The application cannot be reconnected. The value of the option is not passed to the server.

MQCNO_RECONNECT_Q_MGR

The application can be reconnected only to the queue manager with which it originally connected. Use this value if a client can be reconnected, but there is an affinity between the client application and the queue manager with which it originally established a connection. Choose this value if you want a client to automatically reconnect to the standby instance of a highly available queue manager. The value of the option is passed to the server, and can be queried by PCF and MQSC.

Use the options MQCNO_RECONNECT, MQCNO_RECONNECT_DISABLED and MQCNO_RECONNECT_Q_MGR only for client connections. If the options are used for a binding connection, MQCONNX fails with completion code `MQCC_FAILED` and reason code `MQRC_OPTIONS_ERROR`. Automatic client reconnect is not supported by WebSphere MQ classes for Java

Conversation-sharing options

The following options apply only to TCP/IP client connections. For SNA, SPX and NetBios channels, these values are ignored and the channel runs as in previous versions of the product

MQCNO_NO_CONV_SHARING

This option does not permit conversation sharing.

You might use MQCNO_NO_CONV_SHARING in situations where conversations are heavily loaded and, therefore, where contention is a possibility on the server-connection end of the channel instance on which the sharing conversations exist. MQCNO_NO_CNV_SHARING behaves like sharecnv(1) when connected to a channel that supports conversation sharing, and sharecnv(0) when connected to a channel that does not support conversation sharing.

MQCNO_ALL_CONVS_SHARE

This option permits conversation sharing; the application does not place any limit on the number of connections on the channel instance. This option is the default value.

If the application indicates that the channel instance can share, but the *SharingConversations* (SHARECNV) definition on the server-connection end of the channel is set to one, no sharing occurs and no warning is given to the application.

Similarly, if the application indicates that sharing is permitted but the server-connection *SharingConversations* definition is set to zero, no warning is given, and the application exhibits the same behavior as a client in versions of the product earlier than version 7.0; the application setting relating to sharing conversations is ignored.

MQCNO_NO_CONV_SHARING and MQCNO_ALL_CONVS_SHARE are mutually exclusive. If both options are specified on a particular connection, the connection is rejected with a reason code of MQRC_OPTIONS_ERROR.

Channel definition options

The following options control the use of the channel definition structure passed in the MQCNO:

MQCNO_CD_FOR_OUTPUT_ONLY

This option permits channel definition structure in the MQCNO to be used only to return the channel name used on a successful MQCONNX call.

If a valid channel definition structure is not provided, the call fails with the reason code MQRC_CD_ERROR.

If the application is not running as a client, the option is ignored.

The returned channel name can be used on a subsequent MQCONNX call using the MQCNO_USE_CD_SELECTION option to reconnect using the same channel definition. This can be useful when there are multiple applicable channel definitions in the client channel table.

MQCNO_USE_CD_SELECTION

This option permits MQCONNX call to connect using the channel name contained in the channel definition structure passed in the MQCNO.

If the MQSERVER environment variable is set, the channel definition defined by it is used. If MQSERVER is not set, the client channel table is used.

If a channel definition with matching channel name and queue manager name is not found, the call fails with reason code MQRC_Q_MGR_NAME_ERROR.

If a valid channel definition structure is not provided, the call fails with the reason code MQRC_CD_ERROR.

If the application is not running as a client, the option is ignored.

Default option

If you require none of the options described above, you can use the following option:

MQCNO_NONE

No options are specified.

Use MQCNO_NONE to aid program documentation. It is not intended that this option is used with any other MQCNO_* option, but because its value is zero, such use cannot be detected.

SecurityParmsOffset (MQLONG):

SecurityParmsOffset is the offset in bytes of the MQCSP structure from the start of the MQCNO structure. The offset can be positive or negative. This field is an input field, with an initial value of 0.

This field is ignored if *Version* is less than MQCNO_VERSION_5.

The MQCSP structure is defined in "MQCSP - Security parameters" on page 1621.

SecurityParmsPtr (PMQCSP):

SecurityParmsPtr is the address of the MQCSP structure, used to specify a user ID and password for authentication by the authorization service. This field is an input field, and its initial value is a null pointer or null bytes.

This field is ignored if *Version* is less than MQCNO_VERSION_5.

The MQCSP structure is defined in "MQCSP - Security parameters" on page 1621.

SSLConfigOffset (MQLONG):

SSLConfigOffset is the offset in bytes of an MQSCO structure from the start of the MQCNO structure. The offset can be positive or negative. This field is an input field, with an initial value of 0.

Use *SSLConfigOffset* only when the application issuing the MQCONNX call is running as a WebSphere MQ MQI client. For information about how to use this field, see the description of the *SSLConfigPtr* field.

This field is ignored if *Version* is less than MQCNO_VERSION_4.

SSLConfigPtr (PMQSCO):

SSLConfigPtr is an input field. Itsinitial value is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Use *SSLConfigPtr* and *SSLConfigOffset* only when the application issuing the MQCONNX call is running as a WebSphere MQ MQI client and the channel protocol is TCP/IP. If the application is not running as a WebSphere MQ client, or the channel protocol is not TCP/IP, *SSLConfigPtr* and *SSLConfigOffset* are ignored.

By specifying *SSLConfigPtr* or *SSLConfigOffset*, plus either *ClientConnPtr* or *ClientConnOffset*, the application can control the use of SSL for the client connection. When the SSL information is specified in this way, the environment variables MQSSLKEYR and MQSSLCRYP are ignored; any SSL-related information in the client channel definition table (CCDT) is also ignored.

The SSL information can be specified only on:

- The first MQCONNX call of the client process, or
- A subsequent MQCONNX call when all previous SSL/TLS connections to the queue manager have been concluded using MQDISC.

These are the only states in which the process-wide SSL environment can be initialized. If an MQCONNX call is issued specifying SSL information when the SSL environment already exists, the SSL information on the call is ignored and the connection is made using the existing SSL environment; the call returns completion code MQCC_WARNING and reason code MQRC_SSL_ALREADY_INITIALIZED in this case.

You can provide the MQSCO structure in the same way as the MQCD structure, either by specifying an address in *SSLConfigPtr*, or by specifying an offset in *SSLConfigOffset*; see the description of *ClientConnPtr* for details of how to do this. However, you can use no more than one of *SSLConfigPtr* and *SSLConfigOffset*; the call fails with reason code MQRC_SSL_CONFIG_ERROR. if both are nonzero.

Once the MQCONNX call has completed, the MQSCO structure is not referenced again.

This field is ignored if *Version* is less than MQCNO_VERSION_4.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

StrucId (MQCHAR4):

StrucId is always an input field. Its initial value is MQCNO_STRUC_ID.

The value must be:

MQCNO_STRUC_ID
Identifier for connect-options structure.

For the C programming language, the constant MQCNO_STRUC_ID_ARRAY is also defined; this constant has the same value as MQCNO_STRUC_ID, but is an array of characters instead of a string.

Version (MQLONG):

Version is always an input field. Its initial value is MQCNO_VERSION_1.

The value must be one of the following:

MQCNO_VERSION_1
Version-1 connect-options structure.

MQCNO_VERSION_2
Version-2 connect-options structure.

MQCNO_VERSION_3
Version-3 connect-options structure.

MQCNO_VERSION_4
Version-4 connect-options structure.

MQCNO_VERSION_5
Version-5 connect-options structure.

This version of the MQCNO structure extends MQCNO_VERSION_3 on z/OS, and MQCNO_VERSION_4 on all other platforms.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCNO_CURRENT_VERSION
Current version of connect-options structure.

Initial values and language declarations for MQCNO:

Table 136. Initial values of fields in MQCNO for MQCNO

Field name	Name of constant	Value of constant
StrucId	MQCNO_STRUC_ID	'CNO¬'
Version	MQCNO_VERSION_1	1
Options	MQCNO_NONE	0
ClientConnOffset	None	0
ClientConnPtr	None	Null pointer or null bytes
ConnTag	MQCT_NONE	Nulls
SSLConfigPtr	None	Null pointer or null bytes
SSLConfigOffset	None	0
ConnectionId	None	Null pointer or null bytes
SecurityParmsOffset	None	Null pointer or null bytes
SecurityParmsPtr	None	Null pointer or null bytes

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQCNO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQCNO MyCNO = {MQCNO_DEFAULT};
```

C declaration:

```
typedef struct tagMQCNO MQCNO;
struct tagMQCNO {
  MQCHAR4    StrucId;           /* Structure identifier */
  MQLONG     Version;           /* Structure version number */
  MQLONG     Options;           /* Options that control the action of
                                   MQCONNX */
  MQLONG     ClientConnOffset;  /* Offset of MQCD structure for client
                                   connection */
  MQPTR      ClientConnPtr;     /* Address of MQCD structure for client
                                   connection */
  MQBYTE128  ConnTag;           /* Queue-manager connection tag */
  PMQSCO     SSLConfigPtr;      /* Address of MQSCO structure for client
                                   connection */
  MQLONG     SSLConfigOffset;   /* Offset of MQSCO structure for client
                                   connection */
  MQBYTE24   ConnectionId;      /* Unique connection identifier */
  MQLONG     SecurityParmsOffset /* Security fields */
  PMQCSP     SecurityParmsPtr   /* Security parameters */
 };
```

COBOL declaration:

```
**   MQCNO structure
  10 MQCNO.
**    Structure identifier
   15 MQCNO-STRUCID          PIC X(4).
**    Structure version number
   15 MQCNO-VERSION          PIC S9(9) BINARY.
**    Options that control the action of MQCONNX
   15 MQCNO-OPTIONS          PIC S9(9) BINARY.
**    Offset of MQCD structure for client connection
   15 MQCNO-CLIENTCONNOFFSET PIC S9(9) BINARY.
**    Address of MQCD structure for client connection
   15 MQCNO-CLIENTCONNPTR    POINTER.
**    Queue-manager connection tag
   15 MQCNO-CONNTAG          PIC X(128).
**    Address of MQSCO structure for client connection
   15 MQCNO-SSLCONFIGPTR     POINTER.
**    Offset of MQSCO structure for client connection
   15 MQCNO-SSLCONFIGOFFSET  PIC S9(9) BINARY.
**    Unique connection identifier
   15 MQCNO-CONNECTIONID     PIC X(24).
**    Offset of MQCSP structure for security parameters
   15 MQCNO-SECURITYPARMSOFFSET PIC S9(9) BINARY.
**    Address of MQCSP structure for security parameters
   15 MQCNO-SECURITYPARMSPTR POINTER.
```

PL/I declaration:

```
dcl
 1 MQCNO based,
  3 StrucId          char(4),       /* Structure identifier */
  3 Version          fixed bin(31), /* Structure version number */
  3 Options          fixed bin(31), /* Options that control the action
                                       of MQCONNX */
  3 ClientConnOffset fixed bin(31), /* Offset of MQCD structure for
                                       client connection */
  3 ClientConnPtr    pointer,       /* Address of MQCD structure for
                                       client connection */
  3 ConnTag          char(128),     /* Queue-manager connection tag */
  3 SSLConfigPtr     pointer,       /* Address of MQSCO structure for
                                       client connection */
  3 SSLConfigOffset  fixed bin(31), /* Offset of MQSCO structure for
                                       client connection */
  3 ConnectionId     char(24),      /* Unique connection identifier
  3 SecurityParmsOffset fixed bin(31); /* Offset of MQCSP structure for
```

```
                                    security parameters */
  3 SecurityParmsPtr pointer,       /* Address of MQCSP structure for
                                    security parameters */
```

High Level Assembler declaration:

```
MQCNO                   DSECT
MQCNO_STRUCID           DS   CL4    Structure identifier
MQCNO_VERSION           DS   F      Structure version number
MQCNO_OPTIONS           DS   F      Options that control the action of
*                                   MQCONNX
MQCNO_CLIENTCONNOFFSET  DS   F      Offset of MQCD structure for client
*                                   connection
MQCNO_CLIENTCONNPTR     DS   F      Address of MQCD structure for client
*                                   connection
MQCNO_CONNTAG           DS   XL128  Queue-manager connection tag
*
MQCNO_CONNECTIONID      DS   XL24   Unique connection identifier
*
MQCNO_SSLCONFIGOFFSET   DS   F      Offset of MQCSP structure for security
*                                   parameters
MQCNO_SSLCONFIGPTR      DS   F      Address of MQCSP structure for security
*                                   parameters
MQCNO_LENGTH            EQU  *-MQCNO
                        ORG  MQCNO
MQCNO_AREA              DS   CL(MQCNO_LENGTH)
```

Visual Basic declaration:

```
Type MQCNO
  StrucId          As String*4  'Structure identifier'
  Version          As Long      'Structure version number'
  Options          As Long      'Options that control the action of'
                                'MQCONNX'
  ClientConnOffset As Long      'Offset of MQCD structure for client'
                                'connection'
  ClientConnPtr    As MQPTR     'Address of MQCD structure for client'
                                'connection'
  ConnTag          As MQBYTE128 'Queue-manager connection tag'
  SSLConfigPtr     As MQPTR     'Address of MQSCO structure for client'
                                'connection'
  SSLConfigOffset  As Long      'Offset of MQSCO structure for client'
                                'connection'
  ConnectionId     As MQBYTE24  'Unique connection identifier'
  SecurityParmsOffset As Long   'Offset of MQCSP structure for security'
                                'parameters'
  SecurityParmsPtr As MQPTR     'Address of MQCSP structure for security'
                                'parameters'
  End Type
```

MQCSP - Security parameters:

The following table summarizes the fields in the structure.

Table 137. Fields in MQCSP

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
AuthenticationType	Type of authentication	AuthenticationType
Reserved1	Required for pointer alignment on IBM i	Reserved1
CSPUserIdPtr	Address of user ID	CSPUserIdPtr
CSPUserIdOffset	Offset of user ID	CSPUserIdOffset
CSPUserIdLength	Length of user ID	CSPUserIdLength
Reserved2	Required for pointer alignment on IBM i	Reserved2
CSPPasswordPtr	Address of password	CSPPasswordPtr
CSPPasswordOffset	Offset of password	CSPPasswordOffset
CSPPasswordLength	Length of password	CSPPasswordLength

Overview for MQCSP:

Availability: All WebSphere MQ products.

Purpose: The MQCSP structure enables the authorization service to authenticate a user ID and password. You specify the MQCSP connection security parameters structure on an MQCONNX call.

Character set and encoding: Data in MQCSP must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE, respectively.

Fields for MQCSP:

The MQCSP structure contains the following fields; the fields are described in **alphabetical order**:

AuthenticationType (MQLONG):

AuthenticationType is an input field. Its initial value is MQCSP_AUTH_NONE.

This is the type of authentication to perform. Valid values are:

MQCSP_AUTH_NONE
Do not use user ID and password fields.

MQCSP_AUTH_USER_ID_AND_PWD
Authenticate user ID and password fields.

CSPPasswordLength (MQLONG):

This field is the length of the password to be used in authentication.

The maximum length of the password is dependent on the platform, see User IDs. If the length of the password is greater than the maximum length permitted, the authentication request fails with MQRC_NOT_AUTHORIZED.

This field is an input field. The initial value of this field is 0.

CSPPasswordOffset (MQLONG):

This is the offset in bytes of the password to be used in authentication. The offset can be positive or negative.

This is an input field. The initial value of this field is 0.

CSPPasswordPtr (MQPTR):

This is the address in bytes of the password to be used in authentication.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQCNO_VERSION_5.

CSPUserIdLength (MQLONG):

This field is the length of the user ID to be used in authentication.

The maximum length of the user ID is dependent on the platform, see User IDs. If the length of the user ID is greater than the maximum length permitted, the authentication request fails with MQRC_NOT_AUTHORIZED.

This field is an input field. The initial value of this field is 0.

CSPUserIdOffset (MQLONG):

This is the offset in bytes of the user ID to be used in authentication. The offset can be positive or negative.

This is an input field. The initial value of this field is 0.

CSPUserIdPtr (MQPTR):

This is the address in bytes of the user ID to be used in authentication.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQCNO_VERSION_5.

Reserved1 (MQBYTE4):

A reserved field, required for pointer alignment on IBM i.

This is an input field. The initial value of this field is all null.

Reserved2 (MQBYTE8):

A reserved field, required for pointer alignment on IBM i.

This is an input field. The initial value of this field is all null.

StrucId (MQCHAR4):

Structure identifier.

The value must be:

MQCSP_STRUC_ID
Identifier for the security parameters structure.

For the C programming language, the constant MQCSP_STRUC_ID_ARRAY is also defined; this has the same value as MQCSP_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCSPSTRUC_ID.

Version (MQLONG):

Structure version number.

The value must be:

MQCSP_VERSION_1
Version-1 security parameters structure.

The following constant specifies the version number of the current version:

MQCSP_CURRENT_VERSION
Current version of security parameters structure.

This is always an input field. The initial value of this field is MQCSP_VERSION_1.

Initial values and language declarations for MQCSP:

Table 138. Initial values of fields in MQCSP for MQCSP

Field name	Name of constant	Value of constant
StrucId	MQCSP_STRUC_ID	'CSP¬'
Version	MQCSP_CURRENT_VERSION	1
AuthenticationType	None	MQCSP_AUTH_NONE
Reserved1	None	Null string or blanks
CSPUserIdPtr	None	Null pointer or null bytes
CSPUserIdOffset	None	0
CSPUserIdLength	None	0
Reserved2	None	Null string or blanks
CSPPasswordPtr	None	Null pointer or null bytes
CSPPasswordOffset	None	0
CSPPasswordLength	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQCSP_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQCSP MyCSP = {MQCSP_DEFAULT};
```

C declaration:

```
typedef struct tagMQCSP MQCSP;
struct tagMQCSP {
  MQCHAR4    StrucId;            /* Structure identifier */
  MQLONG     Version;            /* Structure version number */
  MQLONG     AuthenticationType; /* Type of authentication */
  MQBYTE4    Reserved1;          /* Required for IBM i pointer
                                    alignment */
  MQPTR      CSPUserIdPtr;       /* Address of user ID */
  MQLONG     CSPUserIdOffset;    /* Offset of user ID */
  MQLONG     CSPUserIdLength;    /* Length of user ID */
  MQBYTE8    Reserved2;          /* Required for IBM i pointer
                                    alignment */
                                 alignment */
  MQPTR      CSPPasswordPtr;     /* Address of password */
  MQLONG     CSPPasswordOffset;  /* Offset of password */
  MQLONG     CSPPasswordLength;  /* Length of password */
  };
```

COBOL declaration:

```
**   MQCSP structure
  10 MQCSP.
**    Structure identifier
   15 MQCSP-STRUCID           PIC X(4).
**    Structure version number
   15 MQCSP-VERSION           PIC S9(9) BINARY.
**    Type of authentication
   15 MQCSP-AUTHENTICATIONTYPE PIC S9(9) BINARY.
**    Required for IBM i pointer alignment
   15 MQCSP-RESERVED1         PIC X(4).
**    Address of user ID
   15 MQCSP-CSPUSERIDPTR      POINTER.
**    Offset of user ID
   15 MQCSP-CSPUSERIDOFFSET   PIC S9(9) BINARY.
**    Length of user ID
   15 MQCSP-CSPUSERIDLENGTH   PIC S9(9) BINARY.
**    Required for IBM i pointer alignment
   15 MQCSP-RESERVED2         PIC X(4).
**    Address of password
   15 MQCSP-CSPPASSWORDPTR    POINTER.
**    Offset of password
   15 MQCSP-CSPPASSWORDOFFSET PIC S9(9) BINARY.
**    Length of password
   15 MQCSP-CSPPASSWORDLENGTH PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQCSP based,
  3 StrucId           char(4),       /* Structure identifier */
  3 Version           fixed bin(31), /* Structure version number */
  3 AuthenticationType fixed bin(31),  /* Type of authentication */
  3 Reserved1         char(4),     /* Required for IBM i pointer
                                      alignment */
  3 CSPUserIdPtr      pointer,      /* Address of user ID */
  3 CSPUserIdOffset   fixed bin(31), /* Offset of user ID */
  3 CSPUserIdLength   fixed bin(31), /* Length of user ID */
  3 Reserved2         char(8),     /* Required for IBM i pointer
                                      alignment */
  3 CSPPasswordPtr    pointer,      /* Address of password */
  3 CSPPasswordOffset fixed bin(31), /* Offset of user ID */
  3 CSPPasswordLength fixed bin(31); /* Length of user ID */
```

Visual Basic declaration:

```
Type MQCSP
  StrucId           As String*4   'Structure identifier'
  Version           As Long       'Structure version number'
  AuthenticationType As Long      'Type of authentication'
  Reserved1         As MQBYTE4    'Required for IBM i pointer'
                                  'alignment'
  CSPUserIdPtr      As MQPTR      'Address of user ID'
  CSPUserIdOffset   As Long       'Offset of user ID'
  CSPUserIdLength   As Long       'Length of user ID'
  Reserved2         As MQBYTE8    'Required for IBM i pointer'
                                  'alignment'
  CSPPasswordPtr    As MQPTR      'Address of password'
  CSPPasswordOffset As Long       'Offset of password'
  CSPPasswordLength As Long       'Length of password'
End Type
```

MQCTLO - Control callback options structure:

The following table summarizes the fields in the structure. Structure specifying the control callback function.

Table 139. Fields in MQCTLO

Field	Description	Topic
StrucID	Structure identifier	StrucID
Version	Structure version number	Version
Options	Options	Options
Reserved	Reserved field	Options
ConnectionArea	Field for callback function to use	ConnectionArea

Overview for MQCTLO:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS, and WebSphere MQ MQI clients connected to these systems. Overview of the MQCTLO structure.

Purpose: The MQCTLO structure is used to specify options relating to a control callbacks function.

The structure is an input and output parameter on the MQCTL call.

Version: The current version of MQCTLO is MQCTLO_VERSION_1.

Character set and encoding: Data in MQCTLO must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQCTLO:

Alphabetic list of fields for the MQCTLO structure.

The MQCTLO structure contains the following fields; the fields are described in alphabetical order:

ConnectionArea (MQPTR):

Control options structure - ConnectionArea field

This is a field that is available for the callback function to use.

The queue manager makes no decisions based on the contents of this field and it is passed unchanged to the ConnectionArea field in the MQCBC structure, which is an input parameter to the callback.

This field is ignored for all operations other than MQOP_START and MQOP_START_WAIT.

This is an input and output field to the callback function. The initial value of this field is a null pointer or null bytes.

Options (MQLONG):

Control options structure - Options field

Options that control the action of MQCTL.

MQCTLO_FAIL_IF_QUIESCING
Force the MQCTL call to fail if the queue manager or connection is in the quiescing state.

Specify MQGMO_FAIL_IF_QUIESCING, in the MQGMO options passed on the MQCB call, to cause notification to message consumers when they are quiescing.

MQCTLO_THREAD_AFFINITY
This option informs the system that the application requires that all message consumers, for the same connection, are called on the same thread. This thread will be used for all invocations of the consumers until the connection is stopped.

Default option: If you do not need any of the options described, use the following option:

MQCTLO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values. MQCTLO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of the *Options* field is MQCTLO_NONE.

Reserved (MQLONG):

This is a reserved field. The value must be zero.

StrucId (MQCHAR4):

Control options structure - StrucId field

This is the structure identifier; the value must be:

MQCTLO_STRUC_ID
Identifier for Control Options structure.

For the C programming language, the constant MQCTLO_STRUC_ID_ARRAY is also defined; this has the same value as MQCTLO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQCTLO_STRUC_ID.

Version (MQLONG):

Control options structure - Version field

This is the structure version number; the value must be:

MQCTLO_VERSION_1
Version-1 Control options structure.

The following constant specifies the version number of the current version:

MQCTLO_CURRENT_VERSION
Current version of Control options structure.

This is always an input field. The initial value of this field is MQCTLO_VERSION_1.

Initial values and language declarations for MQCTLO:

Control options structure - Initial values

Table 140. Initial values of fields in MQCTLO

Field name	Name of constant	Value of constant
StrucId	MQCTLO_STRUC_ID	'CTLO'
Version	MQCTLO_VERSION_1	1
Options	MQCTLO_NONE	Nulls
Reserved	Reserved field	
ConnectionArea	None	Null pointer or null bytes

Notes:

1. In the C programming language, the macro variable MQCTLO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQCTLO MyCTLO = {MQCTLO_DEFAULT};
```

C declaration:

Control Options structure - C language declaration

```
typedef struct tagMQCTLO MQCTLO;
 struct tagMQCTLO {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQLONG    Options;        /* Options that control the action of MQCTL */
  MQLONG    Reserved;       /* Reserved field */

MQPTR     ConnectionArea; /* Connection work area passed to the function */
 };
```

COBOL declaration:

```
** MQCTLO structure
 10  MQCTLO.
** Structure Identifier
 15  MQCTLO-STRUCID                      PIC X(4).
** Structure Version
 15  MQCTLO-VERSION                      PIC S9(9) BINARY.
 ** Options
 15  MQCTLO-OPTIONS                      PIC S9(9) BINARY.
** Reserved
 15  MQCTLO-RESERVED                     PIC S9(9) BINARY.
** ConnectionArea
 15  MQCTLO-CONNECTIONAREA               POINTER
```

PL/I declaration:

```
dcl
 1 MQCTLO based,
  3 StrucId             char(4),       /* Structure identifier */
  3 Version             fixed bin(31), /* Structure version */
  3 Options             fixed bin(31), /* Options */
  3 Reserved            fixed bin(31),
  3 ConnectionArea      pointer;       /* Connection work area */
```

MQDH - Distribution header:

The following table summarizes the fields in the structure.

Table 141. Fields in MQDH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQDH structure plus following records	StrucLength
Encoding	Numeric encoding of data that follows array of MQPMR records	Encoding
CodedCharSetId	Character set identifier of data that follows array of MQPMR records	CodedCharSetId
Format	Format name of data that follows array of MQPMR records	Format
Flags	General flags	Flags
PutMsgRecFields	Flags indicating which MQPMR fields are present	PutMsgRecFields
RecsPresent	Number of object records present	RecsPresent
ObjectRecOffset	Offset of first object record from start of MQDH	ObjectRecOffset

Table 141. Fields in MQDH (continued)

Field	Description	Topic
PutMsgRecOffset	Offset of first put-message record from start of MQDH	PutMsgRecOffset

Overview for MQDH:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQDH structure describes the additional data that is present in a message when that message is a distribution-list message stored on a transmission queue. A distribution-list message is a message that is sent to multiple destination queues. The additional data consists of the MQDH structure followed by an array of MQOR records and an array of MQPMR records.

This structure is used by specialized applications that put messages directly on transmission queues, or that remove messages from transmission queues (for example: message channel agents).

Applications that want to put messages to distribution lists must not use this structure. Instead, they must use the MQOD structure to define the destinations in the distribution list, and the MQPMO structure to specify message properties or receive information about the messages sent to the individual destinations.

Format name: MQFMT_DIST_HEADER.

Character set and encoding: Data in MQDH must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE.

Set the character set and encoding of the MQDH into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQDH structure is at the start of the message data), or
- The header structure that precedes the MQDH structure (all other cases).

Usage: When an application puts a message to a distribution list, and some or all of the destinations are remote, the queue manager prefixes the application message data with the MQXQH and MQDH structures, and places the message on the relevant transmission queue. The data therefore occurs in the following sequence when the message is on a transmission queue:

- MQXQH structure
- MQDH structure plus arrays of MQOR and MQPMR records
- Application message data

Depending on the destinations, the queue manager can generate more than one such message, and place it on different transmission queues. In this case, the MQDH structures in those messages identify different subsets of the destinations defined by the distribution list opened by the application.

An application that puts a distribution-list message directly on a transmission queue must conform to the sequence described above, and must ensure that the MQDH structure is correct. If the MQDH structure is not valid, the queue manager can fail the MQPUT or MQPUT1 call with reason code MQRC_DH_ERROR.

You can store messages on a queue in distribution-list form only if you have defined the queue as being able to support distribution list messages (see the *DistLists* queue attribute described in "Attributes for queues" on page 2135). If an application puts a distribution-list message directly on a queue that does not support distribution lists, the queue manager splits the distribution list message into individual messages,

and places those on the queue instead.

Fields for MQDH:

The MQDH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This is the character set identifier of the data that follows the arrays of MQOR and MQPMR records; it does not apply to character data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. You can use the following special value:

MQCCSI_INHERIT
Inherit character-set identifier of this structure.

Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the MQGET call does not return the value MQCCSI_INHERIT.

You cannot use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This is the numeric encoding of the data that follows the arrays of MQOR and MQPMR records; it does not apply to numeric data in the MQDH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Flags (MQLONG):

You can specify the following flag:

MQDHF_NEW_MSG_IDS
Generate a new message identifier for each destination in the distribution list. Set this only when there are no put-message records present, or when the records are present but they do not contain the *MsgId* field.

Using this flag defers generation of the message identifiers until the moment when the distribution-list message is finally split into individual messages. This minimizes the amount of control information that must flow with the distribution-list message.

When an application puts a message to a distribution list, the queue manager sets MQDHF_NEW_MSG_IDS in the MQDH that it generates when both of the following are true:

- There are no put-message records provided by the application, or the records provided do not contain the *MsgId* field.
- The *MsgId* field in MQMD is MQMI_NONE, or the *Options* field in MQPMO includes MQPMO_NEW_MSG_ID

If no flags are needed, specify the following:

MQDHF_NONE
No flags have been specified. MQDHF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQDHF_NONE.

Format (MQCHAR8):

This is the format name of the data that follows the arrays of MQOD and MQPMR records (whichever occurs last).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

ObjectRecOffset (MQLONG):

This gives the offset in bytes of the first record in the array of MQOR object records containing the names of the destination queues. There are *RecsPresent* records in this array. These records (plus any bytes skipped between the first object record and the previous field) are included in the length given by the *StrucLength* field.

A distribution list must always contain at least one destination, so *ObjectRecOffset* must always be greater than zero.

The initial value of this field is 0.

PutMsgRecFields (MQLONG):

You can specify none or more of the following flags:

MQPMRF_MSG_ID
Message-identifier field is present.

MQPMRF_CORREL_ID
Correlation-identifier field is present.

MQPMRF_GROUP_ID
Group-identifier field is present.

MQPMRF_FEEDBACK
Feedback field is present.

MQPMRF_ACCOUNTING_TOKEN
Accounting-token field is present.

If no MQPMR fields are present, specify the following:

MQPMRF_NONE
No put-message record fields are present. MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQPMRF_NONE.

PutMsgRecOffset (MQLONG):

This gives the offset in bytes of the first record in the array of MQPMR put message records containing the message properties. If present, there are *RecsPresent* records in this array. These records (plus any bytes skipped between the first put message record and the previous field) are included in the length given by the *StrucLength* field.

Put message records are optional; if no records are provided, *PutMsgRecOffset* is zero, and *PutMsgRecFields* has the value MQPMRF_NONE.

The initial value of this field is 0.

RecsPresent (MQLONG):

This is the number of destinations. A distribution list must always contain at least one destination, so *RecsPresent* must always be greater than zero.

The initial value of this field is 0.

StrucId (MQCHAR4):

The value must be:

MQDH_STRUC_ID
Identifier for distribution header structure.

For the C programming language, the constant MQDH_STRUC_ID_ARRAY is also defined; this has the same value as MQDH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQDH_STRUC_ID.

StrucLength (MQLONG):

This is the number of bytes from the start of the MQDH structure to the start of the message data following the arrays of MQOR and MQPMR records. The data occurs in the following sequence:

- MQDH structure
- Array of MQOR records
- Array of MQPMR records
- Message data

The arrays of MQOR and MQPMR records are addressed by offsets contained within the MQDH structure. If these offsets result in unused bytes between one or more of the MQDH structure, the arrays of records, and the message data, those unused bytes must be included in the value of *StrucLength*, but the content of those bytes is not preserved by the queue manager. It is valid for the array of MQPMR records to precede the array of MQOR records.

The initial value of this field is 0.

Version (MQLONG):

The value must be:

MQDH_VERSION_1
Version number for distribution header structure.

The following constant specifies the version number of the current version:

MQDH_CURRENT_VERSION
Current version of distribution header structure.

The initial value of this field is MQDH_VERSION_1.

Initial values and language declarations for MQDH:

Table 142. Initial values of fields in MQDH for MQDH

Field name	Name of constant	Value of constant
StrucId	MQDH_STRUC_ID	'DH¬¬'
Version	MQDH_VERSION_1	1
StrucLength	None	0
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQDHF_NONE	0
PutMsgRecFields	MQPMRF_NONE	0
RecsPresent	None	0
ObjectRecOffset	None	0
PutMsgRecOffset	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQDH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQDH MyDH = {MQDH_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQDH MQDH;
struct tagMQDH {
  MQCHAR4  StrucId;          /* Structure identifier */
  MQLONG   Version;          /* Structure version number */
  MQLONG   StrucLength;      /* Length of MQDH structure plus following
                                MQOR and MQPMR records */
  MQLONG   Encoding;         /* Numeric encoding of data that follows
                                the MQOR and MQPMR records */
  MQLONG   CodedCharSetId;   /* Character set identifier of data that
                                follows the MQOR and MQPMR records */
  MQCHAR8  Format;           /* Format name of data that follows the
                                MQOR and MQPMR records */
  MQLONG   Flags;            /* General flags */
  MQLONG   PutMsgRecFields;  /* Flags indicating which MQPMR fields are
                                present */
  MQLONG   RecsPresent;      /* Number of MQOR records present */
```

```
  MQLONG   ObjectRecOffset;  /* Offset of first MQOR record from start
                                of MQDH */
  MQLONG   PutMsgRecOffset;  /* Offset of first MQPMR record from start
                                of MQDH */
};
```

COBOL declaration:

```
**   MQDH structure
  10 MQDH.
**    Structure identifier
   15 MQDH-STRUCID         PIC X(4).
**    Structure version number
   15 MQDH-VERSION         PIC S9(9) BINARY.
**    Length of MQDH structure plus following MQOR and MQPMR records
   15 MQDH-STRUCLENGTH     PIC S9(9) BINARY.
**    Numeric encoding of data that follows the MQOR and MQPMR records
   15 MQDH-ENCODING        PIC S9(9) BINARY.
**    Character set identifier of data that follows the MQOR and MQPMR
**    records
   15 MQDH-CODEDCHARSETID  PIC S9(9) BINARY.
**    Format name of data that follows the MQOR and MQPMR records
   15 MQDH-FORMAT          PIC X(8).
**    General flags
   15 MQDH-FLAGS           PIC S9(9) BINARY.
**    Flags indicating which MQPMR fields are present
   15 MQDH-PUTMSGRECFIELDS PIC S9(9) BINARY.
**    Number of MQOR records present
   15 MQDH-RECSPRESENT     PIC S9(9) BINARY.
**    Offset of first MQOR record from start of MQDH
   15 MQDH-OBJECTRECOFFSET PIC S9(9) BINARY.
**    Offset of first MQPMR record from start of MQDH
   15 MQDH-PUTMSGRECOFFSET PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQDH based,
  3 StrucId         char(4),       /* Structure identifier */
  3 Version         fixed bin(31), /* Structure version number */
  3 StrucLength     fixed bin(31), /* Length of MQDH structure plus
                                      following MQOR and MQPMR
                                      records */
  3 Encoding        fixed bin(31), /* Numeric encoding of data that
                                      follows the MQOR and MQPMR
                                      records */
  3 CodedCharSetId  fixed bin(31), /* Character set identifier of data
                                      that follows the MQOR and MQPMR
                                      records */
  3 Format          char(8),       /* Format name of data that follows
                                      the MQOR and MQPMR records */
  3 Flags           fixed bin(31), /* General flags */
  3 PutMsgRecFields fixed bin(31), /* Flags indicating which MQPMR
                                      fields are present */
  3 RecsPresent     fixed bin(31), /* Number of MQOR records present */
  3 ObjectRecOffset fixed bin(31), /* Offset of first MQOR record from
                                      start of MQDH */
  3 PutMsgRecOffset fixed bin(31); /* Offset of first MQPMR record from
                                      start of MQDH */
```

Visual Basic declaration:

```
Type MQDH
  StrucId         As String*4 'Structure identifier'
  Version         As Long     'Structure version number'
  StrucLength     As Long     'Length of MQDH structure plus following'
                              'MQOR and MQPMR records'
  Encoding        As Long     'Numeric encoding of data that follows'
                              'the MQOR and MQPMR records'
  CodedCharSetId  As Long     'Character set identifier of data that'
                              'follows the MQOR and MQPMR records'
  Format          As String*8 'Format name of data that follows the'
                              'MQOR and MQPMR records'
  Flags           As Long     'General flags'
  PutMsgRecFields As Long     'Flags indicating which MQPMR fields are'
                              'present'
  RecsPresent     As Long     'Number of MQOR records present'
  ObjectRecOffset As Long     'Offset of first MQOR record from start'
                              'of MQDH'
  PutMsgRecOffset As Long     'Offset of first MQPMR record from start'
                              'of MQDH'
End Type
```

MQDLH - Dead-letter header:

The following table summarizes the fields in the structure.

Table 143. Fields in MQDLH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Reason	Reason message arrived on dead-letter queue	Reason
DestQName	Name of original destination queue	DestQName
DestQMgrName	Name of original destination queue manager	DestQMgrName
Encoding	Numeric encoding of data that follows MQDLH	Encoding
CodedCharSetId	Character set identifier of data that follows MQDLH	CodedCharSetId
Format	Format name of data that follows MQDLH	Format
PutApplType	Type of application that put message on dead-letter queue	PutApplType
PutApplName	Name of application that put message on dead-letter queue	PutApplName
PutDate	Date when message was put on dead-letter queue	PutDate
PutTime	Time when message was put on dead-letter queue	PutTime

Overview for MQDLH:

Availability: All WebSphere MQ platforms.

Purpose: The MQDLH structure describes the information that prefixes the application message data of messages on the dead-letter (undelivered-message) queue. A message can arrive on the dead-letter queue either because the queue manager or message channel agent has redirected it to the queue, or because an application has put the message directly on the queue.

Format name: MQFMT_DEAD_LETTER_HEADER.

Character set and encoding: The fields in the MQDLH structure are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields. These are specified in the header structure that precedes the MQDLH, or in the MQMD structure if the MQDLH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

If you are using the WMQ classes for Java/JMS, and the code page defined in the MQMD is not supported by the Java virtual machine, then the MQDLH is written in the UTF-8 character set.

Usage: Applications that put messages directly on the dead-letter queue must prefix the message data with an MQDLH structure, and initialize the fields with appropriate values. However, the queue manager does not require that an MQDLH structure be present, or that valid values have been specified for the fields.

If a message is too long to put on the dead-letter queue, the application must do one of the following:

- Truncate the message data to fit on the dead-letter queue.
- Record the message on auxiliary storage and place an exception report message on the dead-letter queue indicating this.
- Discard the message and return an error to its originator. If the message is (or might be) a critical message, do this only if it is known that the originator still has a copy of the message; for example, a message received by a message channel agent from a communication channel.

Which of the above is appropriate (if any) depends on the design of the application.

The queue manager performs special processing when a message that is a segment is put with an MQDLH structure at the front; see the description of the MQMDE structure for further details.

Putting messages on the dead-letter queue: When a message is put on the dead-letter queue, the MQMD structure used for the MQPUT or MQPUT1 call must be identical to the MQMD associated with the message (usually the MQMD returned by the MQGET call), with the exception of the following:

- Set the *CodedCharSetId* and *Encoding* fields to whatever character set and encoding are used for fields in the MQDLH structure.
- Set the *Format* field to MQFMT_DEAD_LETTER_HEADER to indicate that the data begins with a MQDLH structure.
- Set the context fields (*AccountingToken*, *ApplIdentityData*, *ApplOriginData*, *PutApplName*, *PutApplType*, *PutDate*, *PutTime*, *UserIdentifier*) by using a context option appropriate to the circumstances:
  - An application putting on the dead-letter queue a message that is not related to any preceding message must use the MQPMO_DEFAULT_CONTEXT option; this causes the queue manager to set all of the context fields in the message descriptor to their default values.
  - A server application putting on the dead-letter queue a messagethat it has just received must use the MQPMO_PASS_ALL_CONTEXT option to preserve the original context information.

– A server application putting on the dead-letter queue a *reply* to a message that it has just received must use the MQPMO_PASS_IDENTITY_CONTEXT option; this preserves the identity information but sets the origin information to be that of the server application.
– A message channel agent putting on the dead-letter queue a messagethat it received from its communication channel must use the MQPMO_SET_ALL_CONTEXT option to preserve the original context information.

In the MQDLH structure itself, set the fields as follows:

- Set the *CodedCharSetId*, *Encoding*, and *Format* fields to the values that describe the data that follows the MQDLH structure, usually the values from the original message descriptor.
- Set the context fields *PutApplType*, *PutApplName*, *PutDate*, and *PutTime* to values appropriate to the application that is putting the message on the dead-letter queue; these values are not related to the original message.
- Set other fields as appropriate.

Ensure that all fields have valid values, and that character fields are padded with blanks to the defined length of the field; do not end the character data prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQDLH structure.

Getting messages from the dead-letter queue: Applications that get messages from the dead-letter queue must verify that the messages begin with an MQDLH structure. The application can determine whether an MQDLH structure is present by examining the *Format* field in the message descriptor MQMD; if the field has the value MQFMT_DEAD_LETTER_HEADER, the message data begins with an MQDLH structure. Be aware also that messages that applications get from the dead-letter queue might be truncated if they were originally too long for the queue.

Fields for MQDLH:

The MQDLH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

CodedCharSetId is the character set identifier of the data that flows through the MQDLH structure (usually the data from the original message); it does not apply to character data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Character data in the data following this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

You cannot use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

DestQMgrName (MQCHAR48):

DestQMgrName is the name of the queue manager that was the original destination for the message.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

DestQName (MQCHAR48):

DestQName is the name of the message queue that was the original destination for the message.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Encoding (MQLONG):

Encoding is the numeric encoding of the data that follows the MQDLH structure (usually the data from the original message); it does not apply to numeric data in the MQDLH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is `0`.

Format (MQCHAR8):

Format is the format name of the data that follows the MQDLH structure (usually the data from the original message).

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those rules for coding the *Format* field in MQMD.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

PutApplName (MQCHAR28):

PutApplName is the name of the application that put the message on the dead-letter (undelivered-message) queue.

The format of the name depends on the *PutApplType* field. The format can vary release to release. See the description of the *PutApplName* field in "MQMD - Message descriptor" on page 1705.

If the queue manager redirects the message to the dead-letter queue, *PutApplName* contains the first 28 characters of the queue-manager name, padded with blanks if necessary.

The length of this field is given by MQ_PUT_APPL_NAME_LENGTH. The initial value of this field is the null string in C, and 28 blank characters in other programming languages.

PutApplType (MQLONG):

PutApplType is the type of application that put the message on the dead-letter (undelivered-message) queue.

This field has the same meaning as the *PutApplType* field in the message descriptor MQMD (see "MQMD - Message descriptor" on page 1705 for details).

If the queue manager redirects the message to the dead-letter queue, *PutApplType* has the value MQAT_QMGR.

The initial value of this field is `0`.

PutDate (MQCHAR8):

PutDate is the date when the message was put on the dead-letter (undelivered-message) queue.

The format used for the date when this field is generated by the queue manager is:

- YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

The length of this field is given by MQ_PUT_DATE_LENGTH. The initial value of this field is the null string in C, and eight blank characters in other programming languages.

PutTime (MQCHAR8):

PutTime is the time when the message was put on the dead-letter (undelivered-message) queue.

The format used for the time when this field is generated by the queue manager is:

- HHMMSSTH

where the characters represent:

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note)

T tenths of a second (0 through 9)

H hundredths of a second (0 through 9)

Note: If the system clock is synchronized to an very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in *PutTime*. This happens when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

The length of this field is given by MQ_PUT_TIME_LENGTH. The initial value of this field is the null string in C, and eight blank characters in other programming languages.

Reason (MQLONG):

The Reason field identifies the reason why the message was placed on the dead-letter queue instead of on the original destination queue.

This identifies the reason why the message was placed on the dead-letter queue instead of on the original destination queue. It should be one of the MQFB_* or MQRC_* values (for example, MQRC_Q_FULL). See the description of the *Feedback* field in "MQMD - Message descriptor" on page 1705 for details of the common MQFB_* values that can occur.

If the value is in the range MQFB_IMS_FIRST through MQFB_IMS_LAST, the actual IMS error code can be determined by subtracting MQFB_IMS_ERROR from the value of the *Reason* field.

Some MQFB_* values occur only in this field. They relate to repository messages, trigger messages, or transmission-queue messages that have been transferred to the dead-letter queue. These are:

MQFB_APPL_CANNOT_BE_STARTED (X'00000109')
An application processing a trigger message cannot start the application named in the *ApplId* field of the trigger message (see "MQTM - Trigger message" on page 1897).

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages.

MQFB_APPL_TYPE_ERROR (X'0000010B')
An application processing a trigger message cannot start the application because the *ApplType* field of the trigger message is not valid (see "MQTM - Trigger message" on page 1897).

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages.

MQFB_BIND_OPEN_CLUSRCVR_DEL (X'00000119')
The message was on the SYSTEM.CLUSTER.TRANSMIT.QUEUE intended for a cluster queue that was opened with the MQOO_BIND_ON_OPEN option, but the remote cluster-receiver channel to be used to transmit the message to the destination queue was deleted before the message could be sent. Because MQOO_BIND_ON_OPEN was specified, only the channel selected when the queue was opened can be used to transmit the message. As this channel is no longer available, the message is placed on the dead-letter queue.

MQFB_NOT_A_REPOSITORY_MSG (X'00000118')
The message is not a repository message.

MQFB_STOPPED_BY_CHAD_EXIT (X'00000115')
The message was stopped by channel auto-definition exit.

MQFB_STOPPED_BY_MSG_EXIT (X'0000010D')
The message was stopped by channel message exit.

MQFB_TM_ERROR (X'0000010A')
The *Format* field in MQMD specifies MQFMT_TRIGGER, but the message does not begin with a valid MQTM structure. For example, the *StrucId* mnemonic eye-catcher might not be valid, the *Version* might not be recognized, or the length of the trigger message might be insufficient to contain the MQTM structure.

On z/OS, the CKTI CICS transaction is an example of an application that processes trigger messages and can generate this feedback code.

MQFB_XMIT_Q_MSG_ERROR (X'0000010F')
A message channel agent has found that a message on the transmission queue is not in the correct format. The message channel agent puts the message on the dead-letter queue using this feedback code.

The initial value of this field is MQRC_NONE.

StrucId (MQCHAR4):

StrucId is the structure identifier.

The value must be:

MQDLH_STRUC_ID
Identifier for dead-letter header structure.

For the C programming language, the constant MQDLH_STRUC_ID_ARRAY is also defined; this has the same value as MQDLH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQDLH_STRUC_ID.

Version (MQLONG):

Version is the structure version number.

The value must be:

MQDLH_VERSION_1
Version number for dead-letter header structure.

The following constant specifies the version number of the current version:

MQDLH_CURRENT_VERSION
Current version of dead-letter header structure.

The initial value of this field is MQDLH_VERSION_1.

Initial values and language declarations for MQDLH:

Table 144. Initial values of fields in MQDLH for MQDLH

Field name	Name of constant	Value of constant
StrucId	MQDLH_STRUC_ID	'DLH¬'
Version	MQDLH_VERSION_1	1
Reason	MQRC_NONE	0
DestQName	None	Null string or blanks
DestQMgrName	None	Null string or blanks
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
PutApplType	None	0
PutApplName	None	Null string or blanks
PutDate	None	Null string or blanks
PutTime	None	Null string or blanks

Table 144. Initial values of fields in MQDLH for MQDLH (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ¬ represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQDLH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: `MQDLH MyDLH = {MQDLH_DEFAULT};`		

C declaration:

```
typedef struct tagMQDLH MQDLH;
struct tagMQDLH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Reason;          /* Reason message arrived on dead-letter
                                (undelivered-message) queue */
  MQCHAR48  DestQName;       /* Name of original destination queue */
  MQCHAR48  DestQMgrName;    /* Name of original destination queue
                                manager */
  MQLONG    Encoding;        /* Numeric encoding of data that follows
                                MQDLH */
  MQLONG    CodedCharSetId;  /* Character set identifier of data that
                                follows MQDLH */
  MQCHAR8   Format;          /* Format name of data that follows
                                MQDLH */
  MQLONG    PutApplType;     /* Type of application that put message on
                                dead-letter (undelivered-message)
                                queue */
  MQCHAR28  PutApplName;     /* Name of application that put message on
                                dead-letter (undelivered-message)
                                queue */
  MQCHAR8   PutDate;         /* Date when message was put on dead-letter
                                (undelivered-message) queue */
  MQCHAR8   PutTime;         /* Time when message was put on the
                                dead-letter (undelivered-message)
                                queue */
};
```

COBOL declaration:

```
**   MQDLH structure
  10 MQDLH.
**    Structure identifier
   15 MQDLH-STRUCID        PIC X(4).
**    Structure version number
   15 MQDLH-VERSION        PIC S9(9) BINARY.
**    Reason message arrived on dead-letter (undelivered-message) queue
   15 MQDLH-REASON         PIC S9(9) BINARY.
**    Name of original destination queue
   15 MQDLH-DESTQNAME      PIC X(48).
**    Name of original destination queue manager
   15 MQDLH-DESTQMGRNAME   PIC X(48).
**    Numeric encoding of data that follows MQDLH
   15 MQDLH-ENCODING       PIC S9(9) BINARY.
**    Character set identifier of data that follows MQDLH
   15 MQDLH-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows MQDLH
   15 MQDLH-FORMAT         PIC X(8).
**    Type of application that put message on dead-letter
```

```
**    (undelivered-message) queue
   15 MQDLH-PUTAPPLTYPE    PIC S9(9) BINARY.
**    Name of application that put message on dead-letter
**    (undelivered-message) queue
   15 MQDLH-PUTAPPLNAME    PIC X(28).
**    Date when message was put on dead-letter (undelivered-message)
**    queue
   15 MQDLH-PUTDATE        PIC X(8).
**    Time when message was put on the dead-letter (undelivered-message)
**    queue
   15 MQDLH-PUTTIME        PIC X(8).
```

PL/I declaration:

```
dcl
 1 MQDLH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Reason         fixed bin(31), /* Reason message arrived on
                                     dead-letter (undelivered-message)
                                     queue */
  3 DestQName      char(48),      /* Name of original destination
                                     queue */
  3 DestQMgrName   char(48),      /* Name of original destination queue
                                     manager */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows MQDLH */
  3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                                     that follows MQDLH */
  3 Format         char(8),       /* Format name of data that follows
                                     MQDLH */
  3 PutApplType    fixed bin(31), /* Type of application that put
                                     message on dead-letter
                                     (undelivered-message) queue */
  3 PutApplName    char(28),      /* Name of application that put
                                     message on dead-letter
                                     (undelivered-message) queue */
  3 PutDate        char(8),       /* Date when message was put on
                                     dead-letter (undelivered-message)
                                     queue */
  3 PutTime        char(8);       /* Time when message was put on the
                                     dead-letter (undelivered-message)
                                     queue */
```

High Level Assembler declaration:

```
MQDLH                 DSECT
MQDLH_STRUCID         DS   CL4   Structure identifier
MQDLH_VERSION         DS   F     Structure version number
MQDLH_REASON          DS   F     Reason message arrived on dead-letter
*                                (undelivered-message) queue
MQDLH_DESTQNAME       DS   CL48  Name of original destination queue
MQDLH_DESTQMGRNAME    DS   CL48  Name of original destination queue
*                                manager
MQDLH_ENCODING        DS   F     Numeric encoding of data that follows
*                                MQDLH
MQDLH_CODEDCHARSETID  DS   F     Character set identifier of data that
*                                follows MQDLH
MQDLH_FORMAT          DS   CL8   Format name of data that follows MQDLH
MQDLH_PUTAPPLTYPE     DS   F     Type of application that put message on
*                                dead-letter (undelivered-message) queue
MQDLH_PUTAPPLNAME     DS   CL28  Name of application that put message on
*                                dead-letter (undelivered-message) queue
MQDLH_PUTDATE         DS   CL8   Date when message was put on
*                                dead-letter (undelivered-message) queue
MQDLH_PUTTIME         DS   CL8   Time when message was put on the
*                                dead-letter (undelivered-message) queue
```

```
*
MQDLH_LENGTH          EQU  *-MQDLH
                      ORG  MQDLH
MQDLH_AREA            DS   CL(MQDLH_LENGTH)
```

Visual Basic declaration:

```
Type MQDLH
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  Reason         As Long      'Reason message arrived on dead-letter'
                              '(undelivered-message) queue'
  DestQName      As String*48 'Name of original destination queue'
  DestQMgrName   As String*48 'Name of original destination queue'
                              'manager'
  Encoding       As Long      'Numeric encoding of data that follows'
                              'MQDLH'
  CodedCharSetId As Long      'Character set identifier of data that'
                              'follows MQDLH'
  Format         As String*8  'Format name of data that follows MQDLH'
  PutApplType    As Long      'Type of application that put message on'
                              'dead-letter (undelivered-message) queue'
  PutApplName    As String*28 'Name of application that put message on'
                              'dead-letter (undelivered-message) queue'
  PutDate        As String*8  'Date when message was put on dead-letter'
                              '(undelivered-message) queue'
  PutTime        As String*8  'Time when message was put on the'
                              'dead-letter (undelivered-message) queue'
End Type
```

MQDMHO - Delete message handle options:

The following table summarizes the fields in the structure.

Table 145. Fields in MQDMHO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options

Overview for MQDMHO:

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The **MQDMHO** structure allows applications to specify options that control how message handles are deleted. The structure is an input parameter on the **MQDLTMH** call.

Character set and encoding: Data in **MQDMHO** must be in the character set of the application and encoding of the application (**MQENC_NATIVE**).

Fields for MQDMHO:

The MQDMHO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

The value must be:

MQDMHO_NONE

No options specified.

This is always an input field. The initial value of this field is **MQDMHO_NONE**.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQDMHO_STRUC_ID

Identifier for delete message handle options structure.

For the C programming language, the constant **MQDMHO_STRUC_ID_ARRAY** is also defined; this has the same value as **MQDMHO_STRUC_ID**, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is **MQDMHO_STRUC_ID**.

Version (MQLONG):

This is the structure version number; the value must be:

MQDMHO_VERSION_1
Version-1 delete message handle options structure.

The following constant specifies the version number of the current version:

MQDMHO_CURRENT_VERSION
Current version of delete message handle options structure.

This is always an input field. The initial value of this field is **MQDMHO_VERSION_1**.

Initial values and language declarations for MQDMHO:

Table 146. Initial values of fields in MQDMHO

Field name	Name of constant	Value of constant
StrucId	MQDMHO_STRUC_ID	'DMHO'
Version	MQDMHO_VERSION_1	1
Options	MQDMHO_NONE	0

Notes:

1. In the C programming language, the macro variable MQDMHO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQDMHO MyDMHO = {MQDMHO_DEFAULT};
```

C declaration:

```
typedef struct tagMQDMHO;
struct tagMQDMHO {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Options;         /* Options that control the action of MQDLTMH */
};
```

COBOL declaration:

```
**   MQDMHO structure
  10 MQDMHO.
**    Structure identifier
   15 MQDMHO-STRUCID       PIC X(4).
**    Structure version number
   15 MQDMHO-VERSION       PIC S9(9) BINARY.
**    Options that control the action of MQDLTMH
   15 MQDMHO-OPTIONS       PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQDMHO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of MQDLTMH */
```

High Level Assembler declaration:

```
MQDMHO                DSECT
MQDMHO_STRUCID        DS   CL4   Structure identifier
MQDMHO_VERSION        DS   F     Structure version number
MQDMHO_OPTIONS        DS   F     Options that control the action of
*                                MQDLTMH
MQDMHO_LENGTH         EQU  *-MQDMHO
MQDMHO_AREA           DS   CL(MQDMHO_LENGTH)
```

MQDMPO - Delete message property options:

The following table summarizes the fields in the structure. MQDMPO structure - delete message property options

Table 147. Fields in MQDMPO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options controlling the action of MQDMPO	Options

Overview for MQDMPO:

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQDMPO structure allows applications to specify options that control how properties of messages are deleted. The structure is an input parameter on the MQDLTMP call.

Character set and encoding: Data in MQDMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQDMPO:

Delete message property options structure - fields

The MQDMPO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Delete message property options structure - Options field

Location options: The following options relate to the relative location of the property compared to the property cursor.

MQDMPO_DEL_FIRST
Deletes the first property that matches the specified name.

MQDMPO_DEL_PROP_UNDER_CURSOR
Deletes the property pointed to by the property cursor; that is the property that was last inquired by using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused. It is also reset when the message handle is specified in the *MsgHandle* field of the MQGMO structure on an MQGET call, or MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established, the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE. If the property pointed to by the property cursor has already been deleted, the call also fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE.

If neither of thees options is required, the following option can be used:

MQDMPO_NONE
No options specified.

This field is always an input field. The initial value of this field is MQDMPO_DEL_FIRST.

StrucId (MQCHAR4):

Delete message property options structure - StrucId field

This is the structure identifier. The value must be:

MQDMPO_STRUC_ID
Identifier for delete message property options structure.

For the C programming language, the constant MQDMPO_STRUC_ID_ARRAY is also defined; this has the same value as MQDMPO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQDMPO_STRUC_ID.

Version (MQLONG):

Delete message property options structure - Version field

This is the structure version number. The value must be:

MQDMPO_VERSION_1
Version number for delete message property options structure.

The following constant specifies the version number of the current version:

MQDMPO_CURRENT_VERSION
Current version of delete message property options structure.

This is always an input field. The initial value of this field is MQDMPO_VERSION_1.

Initial values and language declarations for MQDMPO:

Delete message property options structure - Initial values

Table 148. Initial values of fields in MQDPMO

Field name	Name of constant	Value of constant
StrucId	MQDMPO_STRUC_ID	'DMPO'
Version	MQDMPO_VERSION_1	1
Options	Options that control the action of MQDLTMP	MQDMPO_NONE

Notes:

1. In the C programming language, the macro variable MQDMPO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQDMPO MyDMPO = {MQDMPO_DEFAULT};
   ```

C declaration:

Delete message property options structure - C language declaration

```
typedef struct tagMQDMPO MQDMPO;
struct tagMQDMPO {
  MQCHAR4  StrucId;        /* Structure identifier */
  MQLONG   Version;        /* Structure version number */
  MQLONG   Options;        /* Options that control the action of
                              MQDLTMP */
};
```

COBOL declaration:

Delete message property options structure - COBOL language declaration

```
**   MQDMPO structure
  10 MQDMPO.
**    Structure identifier
   15 MQDMPO-STRUCID               PIC X(4).
**    Structure version number
   15 MQDMPO-VERSION               PIC S9(9) BINARY.
**    Options that control the action of MQDLTMP
   15 MQDMPO-OPTIONS               PIC S9(9) BINARY.
```

PL/I declaration:

Delete message property options structure - PL/I language declaration

```
Dcl
 1 MQDPMO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action
                                     of MQDLTMP */
```

High Level Assembler declaration:

Delete message property options structure - Assembler language declaration

```
MQDMPO                  DSECT
MQDMPO_STRUCID          DS   CL4  Structure identifier
MQDMPO_VERSION          DS   F    Structure version number
MQDMPO_OPTIONS          DS   F    Options that control the
*                                 action of MQDLTMP
MQDMPO_LENGTH           EQU  *-MQDMPO
MQDMPO_AREA             DS   CL(MQDMPO_LENGTH)
```

MQEPH - Embedded PCF header:

The following table summarizes the fields in the structure.

Table 149. Fields in MQEPH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQEPH structure plus the MQCFH and parameter structures that follow it	StrucLength
Encoding	Numeric encoding of data that follows last PCF parameter structure	Encoding
CodedCharSetId	Character set identifier of data that follows last PCF parameter structure	CodedCharSetId
Format	Format name of data that follows last PCF parameter structure	Format
Flags	Flags	Flags
PCFHeader	Programmable command format (PCF) header	PCFHeader

Overview for MQEPH:

Availability: All WebSphere MQ platforms.

Purpose: The MQEPH structure describes the additional data that is present in a message when that message is a programmable command format (PCF) message. The *PCFHeader* field defines the PCF parameters that follow this structure and this allows you to follow the PCF message data with other headers.

Format name: MQFMT_EMBEDDED_PCF

Character set and encoding: Data in MQEPH must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE.

Set the character set and encoding of the MQEPH into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQEPH structure is at the start of the message data), or
- The header structure that precedes the MQEPH structure (all other cases).

Usage: You cannot use MQEPH structures to send commands to the command server or any other queue manager PCF-accepting server.

Similarly, the command server or any other queue manager PCF-accepting server do not generate responses or events containing MQEPH structures.

Fields for MQEPH:

The MQEPH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This is the character set identifier of the data that follows the MQEPH structure and the associated PCF parameters; it does not apply to character data in the MQEPH structure itself.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This is the numeric encoding of the data that follows the MQEPH structure and the associated PCF parameters; it does not apply to character data in the MQEPH structure itself.

The initial value of this field is 0.

Flags (MQLONG):

The following values are available:

MQEPH_NONE
No flags have been specified. MQEPH_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

MQEPH_CCSID_EMBEDDED
The character set of the parameters containing character data is specified individually within the CodedCharSetId field in each structure. The character set of the StrucId and Format fields is defined by the CodedCharSetId field in the header structure that precedes the MQEPH structure, or by the CodedCharSetId field in the MQMD if the MQEPH is at the start of the message.

The initial value of this field is MQEPH_NONE.

Format (MQCHAR8):

This is the format name of the data that follows the MQEPH structure and the associated PCF parameters.

The initial value of this field is MQFMT_NONE.

PCFHeader (MQCFH):
This is the programmable command format (PCF) header, defining the PCF parameters that follow the MQEPH structure. This enables you to follow the PCF message data with other headers.

The PCF header is initially defined with the the following values:

Table 150. Initial values of fields in MQCFH

Field name	Name of constant	Value of constant
Type	MQCFT_NONE	0
StrucLength	MQCFH_STRUC_LENGH	36
Version	MQCFH_VERSION_3	3
StrucLength	None	0
Command	MQCMD_NONE	0

Table 150. Initial values of fields in MQCFH (continued)

Field name	Name of constant	Value of constant
MsgSeqNumber	None	1
Control	MQCFC_LAST	1
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
ParameterCount	None	0

The application must change the Type from MQCFT_NONE to a valid structure type for the use it is making of the embedded PCF header.

StrucId (MQCHAR4):

The value must be:

MQEPH_STRUC_ID
Identifier for distribution header structure.

For the C programming language, the constant MQEPH_STRUC_ID_ARRAY is also defined; this has the same value as MQDH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQEPH_STRUC_ID.

StrucLength (MQLONG):

This is the amount of data preceding the next header structure. It includes:
- The length of the MQEPH header
- The length of all PCF parameters following the header
- Any blank padding following those parameters

StrucLength must be a multiple of 4.

The fixed length part of the structure is defined by MQEPH_STRUC_LENGTH_FIXED.

The initial value of this field is 68.

Version (MQLONG):

The value must be:

MQEPH_VERSION_1
Version number for embedded PCF header structure.

The following constant specifies the version number of the current version:

MQCFH_VERSION_3
Current version of embedded PCF header structure.

The initial value of this field is MQEPH_VERSION_1.

Initial values and language declarations for MQEPH:

Table 151. Initial values of fields in MQEPH for MQEPH

Field name	Name of constant	Value of constant
StrucId	MQEPH_STRUC_ID	'EPH¬'
Version	MQEPH_VERSION_1	1
StrucLength	MQEPH_STRUC_LENGTH_FIXED	68
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQEPH_NONE	0
PCFHeader	Names and values as defined in Table 150 on page 1651	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQEPH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQEPH MyEPH = {MQEPH_DEFAULT};
```

C declaration:

```
typedef struct tagMQEPH MQEPH;
struct tagMQDH {
  MQCHAR4  StrucId;          /* Structure identifier */
  MQLONG   Version;          /* Structure version number */
  MQLONG   StrucLength;      /* Total length of MQEPH including the MQCFH
                                and parameter structures that follow it */
  MQLONG   Encoding;         /* Numeric encoding of data that follows last
                                PCF parameter structure */
  MQLONG   CodedCharSetId;   /* Character set identifier of data that
                                follows last PCF parameter structure */
  MQCHAR8  Format;           /* Format name of data that follows last PCF
                                parameter structure */
  MQLONG   Flags;            /* Flags */
  MQCFH    PCFHeader;        /* Programmable command format header */
 };
```

COBOL declaration:

```
**   MQEPH structure
  10 MQEPH.
**    Structure identifier
   15 MQEPH-STRUCID         PIC X(4).
**    Structure version number
   15 MQEPH-VERSION         PIC S9(9) BINARY.
**    Total length of MQEPH structure including the MQCFH
**    and parameter structures that follow it
   15 MQEPH-STRUCLENGTH     PIC S9(9) BINARY.
**    Numeric encoding of data that follows last
**    PCF structure
   15 MQEPH-ENCODING        PIC S9(9) BINARY.
**    Character set identifier of data that
**    follows last PCF parameter structure
   15 MQEPH-CODEDCHARSETID  PIC S9(9) BINARY.
**    Format name of data that follows last PCF
**    parameter structure
   15 MQEPH-FORMAT          PIC X(8).
```

```
**    Flags
   15 MQEPH-FLAGS           PIC S9(9) BINARY.
**    Programmable command format header
   15 MQEPH-PCFHEADER.
**      Structure type
     20 MQEPH-PCFHEADER-TYPE           PIC S9(9) BINARY.
**      Structure length
     20 MQEPH-PCFHEADER-STRUCLENGTH    PIC S9(9) BINARY.
**      Structure version number
     20 MQEPH-PCFHEADER-VERSION        PIC S9(9) BINARY.
**      Command identifier
     20 MQEPH-PCFHEADER-COMMAND        PIC S9(9) BINARY.
**      Message sequence number
     20 MQEPH-PCFHEADER-MSGSEQNUMBER   PIC S9(9) BINARY.
**      Control options
     20 MQEPH-PCFHEADER-CONTROL        PIC S9(9) BINARY.
**      Completion code
     20 MQEPH-PCFHEADER-COMPCODE       PIC S9(9) BINARY.
**      Reason code qualifying completion code
     20 MQEPH-PCFHEADER-REASON         PIC S9(9) BINARY.
**      Count of parameter structures
     20 MQEPH-PCFHEADER-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQEPH based,
  3 StrucId         char(4),       /* Structure identifier */
  3 Version         fixed bin(31), /* Structure version number */
  3 StrucLength     fixed bin(31), /* Total Length of MQEPH including the
                                      MQCFH and parameter structures that
                                      follow it
  3 Encoding        fixed bin(31), /* Numeric encoding of data that follows
                                      last PCF parameter structure
  3 CodedCharSetId  fixed bin(31), /* Character set identifier of data that
                                      follows last PCF parameter structure
  3 Format          char(8),       /* Format name of data that follows last
                                      PCF parameter structure */
  3 Flags           fixed bin(31), /* Flags */
  3 PCFHeader,                     /* Programmable command format header
   5 Type           fixed bin(31), /* Structure type */
   5 StrucLength    fixed bin(31), /* Structure length */
   5 Version        fixed bin(31), /* Structure version number */
   5 Command        fixed bin(31), /* Command identifier */
   5 MsgseqNumber   fixed bin(31), /* Message sequence number */
   5 Control        fixed bin(31), /* Control options */
   5 CompCode       fixed bin(31), /* Completion code */
   5 Reason         fixed bin(31), /* Reason code qualifying completion code */
   5 ParameterCount fixed bin(31); /* Count of parameter structures */
```

High Level Assembler declaration:

```
MQEPH                          DSECT
MQEPH_STRUCID                  DS   CL4   Structure identifier
MQEPH_VERSION                  DS   F     Structure version number
MQEPH_STRUCLENGTH              DS   F     Total length of MQEPH including the
*                                         MQCFH and parameter structures that
                                          follow it
MQEPH_ENCODING                 DS   F     Numeric encoding of data that follows
*                                         last PCF parameter structure
MQEPH_CODEDCHARSETID           DS   F     Character set identifier of data that
*                                         follows last PCF parameter structure
MQEPH_FORMAT                   DS   CL8   Format name of data that follows last
*                                         PCF parameter structure
MQEPH_FLAGS                    DS   F     Flags
MQEPH_PCFHEADER                DS   0F    Force fullword alignment
MQEPH_PCFHEADER_TYPE           DS   F     Structure type
MQEPH_PCFHEADER_STRUCLENGTH    DS   F     Structure length
```

```
MQEPH_PCFHEADER_VERSION          DS   F     Structure version number
MQEPH_PCFHEADER_COMMAND          DS   F     Command identifier
MQEPH_PCFHEADER_MSGSEQNUMBER     DS   F     Structure length
MQEPH_PCFHEADER_CONTROL          DS   F     Control options
MQEPH_PCFHEADER_COMPCODE         DS   F     Completion code
MQEPH_PCFHEADER_REASON           DS   F     Reason code qualifying completion code
MQEPH_PCFHEADER_PARAMETER COUNT  DS   F     Count of parameter structures
MQEPH_PCFHEADER_LENGTH           EQU  *-MQEPH_PCFHEADER
                                 ORG  MQEPH_PCFHEADER
MQEPH_PCFHEADER_AREA             DS   CL(MQEPH_PCFHEADER_LENGTH)
*
MQEPH_LENGTH                     EQU  *-MQEPH
                                 ORG  MQEPH
MQEPH_AREA                       DS   CL(MQEPH_LENGTH)
```

Visual Basic declaration:

```
Type MQEPH
  StrucId         As String*4 'Structure identifier'
  Version         As Long     'Structure version number'
  StrucLength     As Long     'Total length of MQEPH structure including the MQCFH'
                              'and parameter structures that follow it'
  Encoding        As Long     'Numeric encoding of data that follows last'
                              'PCF parameter structure'
  CodedCharSetId  As Long     'Character set identifier of data that'
                              'follows last PCF parameter structure'
  Format          As String*8 'Format name of data that follows last PCF'
                              'parameter structure'
  Flags           As Long     'Flags'
  PCFHeader       As MQCFH    'Programmable command format header'
 End Type

Global MQEPH_DEFAULT As MQEPH
```

MQGMO - Get-message options:

The following table summarizes the fields in the structure.

Table 152. Fields in MQGMO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options that control the action of MQGET	MQGMO - Options field
WaitInterval	Wait interval	WaitInterval
Signal1	Signal	Signal1
Signal2	Signal identifier	Signal2
ResolvedQName	Resolved name of destination queue	ResolvedQName
Note: The remaining fields are ignored if *Version* is less than MQGMO_VERSION_2.		
MatchOptions	Options controlling selection criteria used for MQGET	MatchOptions
GroupStatus	Flag indicating whether message retrieved is in a group	GroupStatus
SegmentStatus	Flag indicating whether message retrieved is a segment of a logical message	SegmentStatus
Segmentation	Flag indicating whether further segmentation is allowed for the message retrieved	Segmentation
Reserved1	Reserved	Reserved1

Table 152. Fields in MQGMO (continued)

Field	Description	Topic
Note: The remaining fields are ignored if *Version* is less than MQGMO_VERSION_3.		
MsgToken	Message token	MsgToken
ReturnedLength	Length of message data returned (bytes)	ReturnedLength
Note: The remaining fields are ignored if *Version* is less than MQGMO_VERSION_4.		
Reserved2	Reserved	Reserved2
MsgHandle	The handle to a message that is to be populated with the properties of the message being retrieved from the queue.	MsgHandle

Overview for MQGMO:

Availability: All WebSphere MQ platforms.

Purpose: The MQGMO structure allows the application to control how messages are removed from queues. The structure is an input/output parameter on the MQGET call.

Version: The current version of MQGMO is MQGMO_VERSION_4. Certain fields are available only in certain versions of MQGMO. If you need to port applications between several environments, you must ensure that the version of MQGMO is consistent across all environments. Fields that exist only in particular versions of the structure are identified as such in "MQGMO - Get-message options" on page 1655 and in the field descriptions.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQGMO that is supported by the environment, but with the initial value of the *Version* field set to MQGMO_VERSION_1. To use fields that are not present in the version-1 structure, set the *Version* field to the version number of the version required.

Character set and encoding: Data in MQGMO must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQGMO:

The MQGMO structure contains the following fields; the fields are described in **alphabetical order**:

GroupStatus (MQCHAR):

This flag indicates whether the message retrieved is in a group.

It has one of the following values:

MQGS_NOT_IN_GROUP
Message is not in a group.

MQGS_MSG_IN_GROUP
Message is in a group, but is not the last in the group.

MQGS_LAST_MSG_IN_GROUP
Message is the last in the group.

This is also the value returned if the group consists of only one message.

This is an output field. The initial value of this field is MQGS_NOT_IN_GROUP. This field is ignored if *Version* is less than MQGMO_VERSION_2.

MatchOptions (MQLONG):

These options allow the application to choose which fields in the *MsgDesc* parameter to use to select the message returned by the MQGET call. The application sets the required options in this field, and then sets the corresponding fields in the *MsgDesc* parameter to the values required for those fields. Only messages that have those values in the MQMD for the message are candidates for retrieval using that *MsgDesc* parameter on the MQGET call. Fields for which the corresponding match option is *not* specified are ignored when selecting the message to be returned. If you specify no selection criteria on the MQGET call (that is, *any* message is acceptable), set *MatchOptions* to MQMO_NONE.

- On z/OS, the selection criteria that can be used might be restricted by the type of index used for the queue. See the *IndexType* queue attribute for further details.

If you specify MQGMO_LOGICAL_ORDER, only certain messages are eligible for return by the next MQGET call:

- If there is no current group or logical message, only messages that have *MsgSeqNumber* equal to 1 and *Offset* equal to 0 are eligible for return. In this situation, you can use one or more of the following match options to select which of the eligible messages is returned:
  - MQMO_MATCH_MSG_ID
  - MQMO_MATCH_CORREL_ID
  - MQMO_MATCH_GROUP_ID
- If there *is* a current group or logical message, only the next message in the group or next segment in the logical message is eligible for return, and this cannot be altered by specifying MQMO_* options.

In both of the above cases, you can specify match options that do not apply, but the value of the relevant field in the *MsgDesc* parameter must match the value of the corresponding field in the message to be returned; the call fails with reason code MQRC_MATCH_OPTIONS_ERROR is this condition is not satisfied.

MatchOptions is ignored if you specify either MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR.

Getting messages based on message property is not done using match options; for more information, see "SelectionString (MQCHARV)" on page 1778 .

You can specify one or more of the following match options:

MQMO_MATCH_MSG_ID
The message to be retrieved must have a message identifier that matches the value of the *MsgId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the *MsgId* field in the *MsgDesc* parameter is ignored, and any message identifier will match.

Note: The message identifier MQMI_NONE is a special value that matches *any* message identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_MSG_ID with MQMI_NONE is the same as *not* specifying MQMO_MATCH_MSG_ID.

MQMO_MATCH_CORREL_ID
The message to be retrieved must have a correlation identifier that matches the value of the *CorrelId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message identifier).

If you omit this option, the *CorrelId* field in the *MsgDesc* parameter is ignored, and any correlation identifier will match.

Note: The correlation identifier MQCI_NONE is a special value that matches *any* correlation identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_CORREL_ID with MQCI_NONE is the same as *not* specifying MQMO_MATCH_CORREL_ID.

MQMO_MATCH_GROUP_ID
The message to be retrieved must have a group identifier that matches the value of the *GroupId* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the *GroupId* field in the *MsgDesc* parameter is ignored, and any group identifier will match.

Note: The group identifier MQGI_NONE is a special value that matches *any* group identifier in the MQMD for the message. Therefore, specifying MQMO_MATCH_GROUP_ID with MQGI_NONE is the same as *not* specifying MQMO_MATCH_GROUP_ID.

MQMO_MATCH_MSG_SEQ_NUMBER
The message to be retrieved must have a message sequence number that matches the value of the *MsgSeqNumber* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the group identifier).

If you omit this option, the *MsgSeqNumber* field in the *MsgDesc* parameter is ignored, and any message sequence number will match.

MQMO_MATCH_OFFSET
The message to be retrieved must have an offset that matches the value of the *Offset* field in the *MsgDesc* parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message sequence number).

If you omit this option is not specified, the *Offset* field in the *MsgDesc* parameter is ignored, and any offset will match.

- This option is not supported on z/OS.

MQMO_MATCH_MSG_TOKEN
The message to be retrieved must have a message token that matches the value of the *MsgToken* field in the MQGMO structure specified on the MQGET call.

You can specify this option for all local queues. If you specify it for a queue that has an *IndexType* of MQIT_MSG_TOKEN (a WLM-managed queue), you can specify no other match options with MQMO_MATCH_MSG_TOKEN.

You cannot specify MQMO_MATCH_MSG_TOKEN with MQGMO_WAIT or MQGMO_SET_SIGNAL. If the application wants to wait for a message to arrive on a queue that has an *IndexType* of MQIT_MSG_TOKEN, specify MQMO_NONE.

If you omit this option, the *MsgToken* field in MQGMO is ignored, and any message token will match.

If you specify none of the options described, you can use the following option:

MQMO_NONE
Use no matches in selecting the message to be returned; all messages on the queue are eligible for retrieval (but subject to control by the MQGMO_ALL_MSGS_AVAILABLE, MQGMO_ALL_SEGMENTS_AVAILABLE, and MQGMO_COMPLETE_MSG options).

MQMO_NONE aids program documentation. It is not intended that this option be used with any other MQMO_* option, but as its value is zero, such use cannot be detected.

This is an input field. The initial value of this field is MQMO_MATCH_MSG_ID with MQMO_MATCH_CORREL_ID. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Note: The initial value of the *MatchOptions* field is defined for compatibility with earlier MQSeries queue managers. However, when reading a series of messages from a queue without using selection criteria, this initial value requires the application to reset the *MsgId* and *CorrelId* fields to MQMI_NONE and MQCI_NONE before each MQGET call. Avoid the need to reset *MsgId* and *CorrelId* by setting *Version* to MQGMO_VERSION_2, and *MatchOptions* to MQMO_NONE.

MsgHandle (MQHMSG):

If the MQGMO_PROPERTIES_AS_Q_DEF option is specified and the PropertyControl queue attribute is not set to MQPROP_FORCE_MQRFH2 then this is the handle to a message which will be populated with the properties of the message being retrieved from the queue. The handle is created by an MQCRTMH call. Any properties already associated with the handle will be cleared before retrieving a message.

The following value can also be specified:
MQHM_NONE
No message handle supplied.

No message descriptor is required on the MQGET call if a valid message handle is supplied and used on output to contain the message properties, the message descriptor associated with the message handle is used for input fields.

If a message descriptor is specified on the MQGET call, it always takes precedence over the message descriptor associated with a message handle.

If MQGMO_PROPERTIES_FORCE_MQRFH2 is specified, or the MQGMO_PROPERTIES_AS_Q_DEF is specified and the PropertyControl queue attribute is MQPROP_FORCE_MQRFH2 then the call fails with reason code MQRC_MD_ERROR when no message descriptor parameter is specified.

On return from the MQGET call, the properties and message descriptor associated with this message handle are updated to reflect the state of the message retrieved (as well as the message descriptor if one was supplied on the MQGET call). The properties of the message can then be inquired using the MQINQMP call.

Except for message descriptor extensions, when present, a property that can be inquired with the MQINQMP call is not contained in the message data; if the message on the queue contained properties in the message data these are removed from the message data before the data is returned to the application.

If no message handle is provided or Version is less than MQGMO_VERSION_4 then you must supply a valid message descriptor on the MQGET call. Any message properties (except those contained in the message descriptor) are returned in the message data subject to the value of the property options in the MQGMO structure and the PropertyControl queue attribute.

This is an always an input field. The initial value of this field is MQHM_NONE. This field is ignored if Version is less than MQGMO_VERSION_4.

MsgToken (MQBYTE16):

MsgToken field - MQGMO structure. This field is used by the queue manager to uniquely identify a message.

This is a byte string that is generated by the queue manager to identify a message uniquely on a queue. The message token is generated when the message is first placed on the queue manager, and remains with the message until the message is permanently removed from the queue manager, unless the queue manager is restarted.

When the message is removed from the queue, the *MsgToken* that identified that instance of the message is no longer valid, and is never reused. If the queue manager is restarted, the *MsgToken* that identified a message on the queue before restart might not be valid after restart. However, the *MsgToken* is never reused to identify a different message instance. The *MsgToken* is generated by the queue manager and is not visible to any external application.

When a message is returned by a call to MQGET where a Version 3 or higher MQGMO is supplied, the *MsgToken* identifying the message on the queue is returned in the MQGMO by the queue manager. There is one exception to this: when the message is being removed from the queue outside syncpoint, the queue manager might not return a *MsgToken* because it is not useful to identify the returned message on a subsequent MQGET call. Applications should only use *MsgToken* to refer to the message on subsequent MQGET calls.

If a *MsgToken* is supplied and the *MatchOption* MQMO_MATCH_MSG_TOKEN is specified and neither MQGMO_MSG_UNDER_CURSOR nor MQGMO_BROWSE_MSG_UNDER_CURSOR is specified, only the message identified by that *MsgToken* can be returned. The option is valid on all local queues regardless of INDXTYPE, and on z/OS you must use INDXTYPE(MSGTOKEN) only on Workload Manager (WLM) queues.

Any other *MatchOptions* specified are checked, and if they do not match, MQRC_NO_MSG_AVAILABLE is returned. If MQGMO_BROWSE_NEXT is coded with MQMO_MATCH_MSG_TOKEN, the message identified by the *MsgToken* is returned only if it is beyond the browse-cursor for the calling handle.

If MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR is specified, MQMO_MATCH_MSG_TOKEN is ignored.

MQMO_MATCH_MSG_TOKEN is not valid with the following get message options:

- MQGMO_WAIT
- MQGMO_SET_SIGNAL

For an MQGET call specifying MQMO_MATCH_MSG_TOKEN, an MQGMO of version 3 or later must be supplied to the call, otherwise MQRC_WRONG_GMO_VERSION is returned.

If the *MsgToken* is not valid at this time, MQCC_FAILED with MQRC_NO_MSG_AVAILABLE is returned, unless there is another error.

Options (MQLONG):

MQGMO options control the action of MQGET. You can specify zero or more of the options. If you need more than one optional value:

- Add the values (do not add the same constant more than once), or
- Combine the values using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

Wait options: The following options relate to waiting for messages to arrive on the queue:

MQGMO_WAIT
The application waits until a suitable message arrives. The maximum time that the application waits is specified in *WaitInterval*.

Important: There is no wait, or delay, if a suitable message is available immediately.

If MQGET requests are inhibited, or MQGET requests become inhibited while waiting, the wait is canceled. The call completes with MQCC_FAILED and reason code MQRC_GET_INHIBITED, regardless of whether there are suitable messages on the queue.

You can use MQGMO_WAIT with the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT options.

If several applications are waiting on the same shared queue, the following rules select which application is activated when a suitable message arrives:

Table 153. Rules for activating MQGET calls on a shared queue.

Number of MQGET calls waiting to be activated		**Result**
With a BROWSE option	**Without a BROWSE option(An MQGET call specifying the MQGMO_LOCK option is treated as a nonbrowse call.)**	
None	One or more	One MQGET call without a BROWSE option is activated.
One or more	None	All MQGET calls with a BROWSE option are activated.
One or more	One or more	One MQGET call without a BROWSE option is activated. The number of MQGET calls with a BROWSE option that are activated is unpredictable.

If more than one MQGET call without a BROWSE option is waiting on the same queue, only one is activated. The queue manager attempts to give priority to waiting calls in the following order:

1. Specific get-wait requests that can be satisfied only by certain messages, for example, ones with a specific *MsgId* or *CorrelId* (or both).
2. General get-wait requests that can be satisfied by any message.

Note:

- Within the first category, no additional priority is given to more specific get-wait requests. For example, requests that specify both *MsgId* and *CorrelId*.
- Within either category, it cannot be predicted which application is selected. In particular, the application waiting longest is not necessarily the one selected.
- Path length, and priority-scheduling considerations of the operating system, can mean that a waiting application of lower operating system priority than expected retrieves the message.
- It can also happen that an application that is not waiting retrieves the message in preference to one that is.

On z/OS, the following points apply:

- If you want the application to proceed with other work while waiting for the message to arrive, consider using the signal option (MQGMO_SET_SIGNAL) instead. However the signal option is environment-specific; applications that you to port between different environments must not use it.
- If there is more than one MQGET call waiting for the same message, with a mixture of wait and signal options, each waiting call is considered equally. It is an error to specify MQGMO_SET_SIGNAL with MQGMO_WAIT. It is also an error to specify this option with a queue handle for which a signal is outstanding.
- If you specify MQGMO_WAIT or MQGMO_SET_SIGNAL for a queue that has an *IndexType* of MQIT_MSG_TOKEN, no selection criteria are permitted. This means that:
  - If you are using a version-1 MQGMO, set the *MsgId* and *CorrelId* fields in the MQMD specified on the MQGET call to MQMI_NONE and MQCI_NONE.
  - If you are using a version-2 or later MQGMO, set the *MatchOptions* field to MQMO_NONE.
- For an MQGET call on a shared queue and the call is a browse request, or a destructive get of a group message, and neither *MsgId* or *CorrelId* are to be matched, the MQGET call is reissued every 200 milliseconds until a suitable message arrives on the queue or the wait interval expires.

  This method causes an unexpected processing overhead and is not an efficient method of message retrieval when messages are added infrequently. To avoid this overhead for the browse case, specify *MsgId* (if non-indexed or indexed by *MsgId*) or *CorrelId* (if indexed by *CorrelId*) matching on the MQGET call.

MQGMO_WAIT is ignored if specified with MQGMO_BROWSE_MSG_UNDER_CURSOR or MQGMO_MSG_UNDER_CURSOR; no error is raised.

MQGMO_NO_WAIT
The application does not wait if no suitable message is available. MQGMO_NO_WAIT is the opposite of the MQGMO_WAIT. MQGMO_NO_WAIT is defined to aid program documentation. It is the default if neither is specified.

MQGMO_SET_SIGNAL
Use this option with the *Signal1* and *Signal2* fields. It allows applications to proceed with other work while waiting for a message to arrive. It also allows (if suitable operating system facilities are available) applications to wait for messages arriving on more than one queue.

Note: The MQGMO_SET_SIGNAL option is environment-specific; do not use it for applications that you want to port.

In two circumstances, the call completes in the same way as if this option had not been specified:

1. If a currently available message satisfies the criteria specified in the message descriptor.
2. If a parameter error or other synchronous error is detected.

If no message satisfying the criteria specified in the message descriptor is currently available, control returns to the application without waiting for a message to arrive. The *CompCode* and *Reason* parameters are set to MQCC_WARNING and MQRC_SIGNAL_REQUEST_ACCEPTED. Other output fields in the message descriptor and the output parameters of the MQGET call are not set. When a suitable message arrives later, the signal is delivered by posting the ECB.

The caller must then reissue the MQGET call to retrieve the message. The application can wait for this signal, using functions provided by the operating system.

If the operating system provides a multiple wait mechanism, you can use it to wait for a message arriving on any one of several queues.

If a nonzero *WaitInterval* is specified, the signal is delivered after the wait interval expires. The queue manager can also cancel the wait, in which case the signal is delivered.

More than one MQGET call can set a signal for the same message. The order in which applications are activated is the same as described for MQGMO_WAIT.

If more than one MQGET call is waiting for the same message, each waiting call is considered equally. The calls can include a mixture of wait and signal options.

Under certain conditions the MQGET call can retrieve a message, and a signal resulting from the arrival of the same message can be delivered. When a signal is delivered, an application must be prepared for no message to be available.

A queue handle can have no more than one signal request outstanding.

This option is not valid with any of the following options:

- MQGMO_UNLOCK
- MQGMO_WAIT

For an MQGET call on a shared queue and the call is a browse request, or a destructive get of a group message, and neither *MsgId* or *CorrelId* are to be matched, the user's signal ECB is posted MQEC_MSG_ARRIVED after 200 milliseconds.

This occurs, even though a suitable message might not have arrived on the queue, until the wait interval has expired, when the queue is posted with MQEC_WAIT_INTERVAL_EXPIRED. When MQEC_MSG_ARRIVED is posted, you must reissue a second MQGET call to retrieve the message, if one is available.

This technique is used to ensure that you are informed in a timely manner of a message arrival, but can appear as an unexpected processing overhead when compared with a similar call sequence on a nonshared queue.

This is not an efficient method of message retrieval when messages are added infrequently. To avoid this overhead for the browse case, specify *MsgId* (if non-indexed or indexed by *MsgId*) or *CorrelId* (if indexed by *CorrelId*) matching on the MQGET call.

This option is supported on z/OS only.

MQGMO_FAIL_IF_QUIESCING
Force the MQGET call to fail if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQGET call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

If this option is specified with MQGMO_WAIT or MQGMO_SET_SIGNAL, and the wait or signal is outstanding at the time the queue manager enters the quiescing state:

- The wait is canceled and the call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR_QUIESCING or MQRC_CONNECTION_QUIESCING.
- The signal is canceled with an environment-specific signal completion code.

  On z/OS, the signal completes with event completion code MQEC_Q_MGR_QUIESCING or MQEC_CONNECTION_QUIESCING.

If MQGMO_FAIL_IF_QUIESCING is not specified and the queue manager or connection enters the quiescing state, the wait or signal is not canceled.

Sync point options: The following options relate to the participation of the MQGET call within a unit of work:

MQGMO_SYNCPOINT
The request is to operate within the normal unit-of-work protocols. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

You can leave MQGMO_SYNCPOINT and MQGMO_NO_SYNCPOINT unset. In which case, the inclusion of the get request in unit-of-work protocols is determined by the environment running the queue

manager. It is not determined by the environment running the application. On z/OS, the get request is within a unit of work. In all other environments, the get request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT explicitly.

This option is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_LOCK
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

MQGMO_SYNCPOINT_IF_PERSISTENT

The request is to operate within the normal unit-of-work protocols, but *only* if the message retrieved is persistent. A persistent message has the value MQPER_PERSISTENT in the *Persistence* field in MQMD.

- If the message is persistent, the queue manager processes the call as though the application had specified MQGMO_SYNCPOINT.
- If the message is not persistent, the queue manager processes the call as though the application had specified MQGMO_NO_SYNCPOINT.

This option is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_COMPLETE_MSG
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT
- MQGMO_UNLOCK

This option is supported in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, and Linux, plus WebSphere MQ MQI clients connected to these systems.

MQGMO_NO_SYNCPOINT

The request is to operate outside the normal unit-of-work protocols. If you get a message without a browse option, it is deleted from the queue immediately. The message cannot be made available again by backing out the unit of work.

This option is assumed if you specify MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT.

You can leave MQGMO_SYNCPOINT and MQGMO_NO_SYNCPOINT unset. In which case, the inclusion of the get request in unit-of-work protocols is determined by the environment running the queue manager. It is not determined by the environment running the application. On z/OS, the get request is within a unit of work. In all other environments, the get request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT explicitly.

This option is not valid with any of the following options:

- MQGMO_MARK_SKIP_BACKOUT

- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT

MQGMO_MARK_SKIP_BACKOUT
Back out a unit of work without reinstating on the queue the message that was marked with this option.

This option is supported only on z/OS.
If this option is specified, MQGMO_SYNCPOINT must also be specified. MQGMO_MARK_SKIP_BACKOUT is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_LOCK
- MQGMO_NO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

Note: On IMS and CICS, you might have to issue an extra WebSphere MQ call after backing out a unit of work containing a message marked with MQGMO_MARK_SKIP_BACKOUT. You must issue a WebSphere MQ call before you commit the new unit of work containing the marked message. The call can be any WebSphere MQ call you like.

1. On IMS, if you have not applied IMS APAR PN60855 and you are running an IMS MPP or BMP application.
2. On CICS, if you are running any application.

In both cases, issue any WebSphere MQ call before committing the new unit of work containing the backed out message.

Note: Within a unit of work, there can be only one get request marked as skipping backout, as well as none or several unmarked get requests.
If an application backs out of a unit of work, a message that was retrieved using MQGMO_MARK_SKIP_BACKOUT is not restored to its previous state. Other resource updates are backed out. The message is treated as if it had been retrieved in a new unit of work started by the backout request. The message is retrieved without the MQGMO_MARK_SKIP_BACKOUT option. MQGMO_MARK_SKIP_BACKOUT is useful if, after some resources have been changed, it becomes apparent that the unit of work cannot complete successfully. If you omit this option, backing out the unit of work reinstates the message on the queue. The same sequence of events occurs again, when the message is next retrieved.
However, if you specify MQGMO_MARK_SKIP_BACKOUT on the original MQGET call, backing out the unit of work backs out the updates to the other resources. The message is treated as if it had been retrieved under a new unit of work. The application can perform appropriate error handling. It can send a report message to the sender of the original message, or place the original message on the dead-letter queue. It can then commit the new unit of work. Committing the new unit of work removes the message permanently from the original queue.
MQGMO_MARK_SKIP_BACKOUT marks a single physical message. If the message belongs to a message group, the other messages in the group are not marked. Similarly, if the marked message is a segment of a logical message, the other segments in the logical message are not marked.
Any message in a group can be marked, but if messages are retrieved using MQGMO_LOGICAL_ORDER, it is advantageous to mark the first message in the group. If the unit of work is backed out, the first (marked) message is moved to the new unit of work. The second and later messages in the group are reinstated on the queue. The messages left on the queue cannot be retrieved by another application using MQGMO_LOGICAL_ORDER. The first message in the group is no longer on the queue. However, the application that backed the unit of work out can retrieve the second and later messages into the new unit of work using the MQGMO_LOGICAL_ORDER

option. The first message has been retrieved already.
Occasionally you might need to back out the new unit of work. For example, because the dead-letter queue is full and the message must not be discarded. Backing out the new unit of work reinstates the message on the original queue, which prevents the message being lost. However, in this situation processing cannot continue. After backing out the new unit of work, the application must inform the operator or administrator that there is an unrecoverable error, and then finish.
MQGMO_MARK_SKIP_BACKOUT only works if the unit of work containing the get request is interrupted by the application backing it out. If the unit of work containing the get request is backed out because the transaction or system fails, MQGMO_MARK_SKIP_BACKOUT is ignored. Any message retrieved using this option is reinstated on the queue in the same way as messages retrieved without this option.

Browse options: The following options relate to browsing messages on the queue:

MQGMO_BROWSE_FIRST
When a queue is opened with the MQOO_BROWSE option, a browse cursor is established, positioned logically before the first message on the queue. You can then use MQGET calls specifying the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option to retrieve messages from the queue nondestructively. The browse cursor marks the position, within the messages on the queue, from which the next MQGET call with MQGMO_BROWSE_NEXT searches for a suitable message.

MQGMO_BROWSE_FIRST is not valid with any of the following options:

- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.

An MQGET call with MQGMO_BROWSE_FIRST ignores the previous position of the browse cursor. The first message on the queue that satisfies the conditions specified in the message descriptor is retrieved. The message remains on the queue, and the browse cursor is positioned on this message.

After this call, the browse cursor is positioned on the message that has been returned. The message might be removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued. In this case, the browse cursor remains at the position in the queue that the message occupied, even though that position is now empty.

Use the MQGMO_MSG_UNDER_CURSOR option with a non-browse MQGET call, to remove the message from the queue.

The browse cursor is not moved by a non-browse MQGET call, even if using the same *Hobj* handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.

Specify the MQGMO_LOCK option with this option, to lock the message that is browsed.

You can specify MQGMO_BROWSE_FIRST with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.

If you specify MQGMO_LOGICAL_ORDER, the messages are browsed in logical order. If you omit that option, the messages are browsed in physical order. If you specify MQGMO_BROWSE_FIRST, you can

switch between logical order and physical order. Subsequent MQGET calls using MQGMO_BROWSE_NEXT browse the queue in the same order as the most recent call that specified MQGMO_BROWSE_FIRST for the queue handle.

The queue manager retains two sets of group and segment information for MQGET calls. The group and segment information for browse calls are retained separately from the information for calls that remove messages from the queue. If you specify MQGMO_BROWSE_FIRST, the queue manager ignores the group and segment information for browsing. It scans the queue as though there were no current group and no current logical message. If the MQGET call is successful, completion code MQCC_OK or MQCC_WARNING, the group and segment information for browsing is set to that of the message returned. If the call fails, the group and segment information remain the same as they were before the call.

MQGMO_BROWSE_NEXT

Advance the browse cursor to the next message on the queue that satisfies the selection criteria specified on the MQGET call. The message is returned to the application, but remains on the queue.

MQGMO_BROWSE_NEXT is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.
MQGMO_BROWSE_NEXT behaves the same way as MQGMO_BROWSE_FIRST, if it is the first call to browse a queue, after the queue has been opened for browse.
The message under cursor might be removed from the queue before the next MQGET call with MQGMO_BROWSE_NEXT is issued. The browse cursor logically remains at the position in the queue that the message occupied, even though that position is now empty.
Messages are stored on the queue in one of two ways:

- FIFO within priority (MQMDS_PRIORITY), or
- FIFO *regardless* of priority (MQMDS_FIFO)

The *MsgDeliverySequence* queue attribute indicates which method applies (see "Attributes for queues" on page 2135 for details).
A queue might have a *MsgDeliverySequence* of MQMDS_PRIORITY. A message arrives on the queue that is of a higher priority than the one currently pointed to by the browse cursor. In which case, the higher priority message is not found during the current sweep of the queue using MQGMO_BROWSE_NEXT. It can be found only after the browse cursor has been reset with MQGMO_BROWSE_FIRST, or by reopening the queue.
The MQGMO_MSG_UNDER_CURSOR option can be used with a non-browse MQGET call if required, to remove the message from the queue.
The browse cursor is not moved by non-browse MQGET calls using the same *Hobj* handle.
Specify the MQGMO_LOCK option with this option to lock the message that is browsed.
You can specify MQGMO_BROWSE_NEXT with any valid combination of the MQGMO_* and MQMO_* options that control the processing of messages in groups and segments of logical messages.
If you specify MQGMO_LOGICAL_ORDER, the messages are browsed in logical order. If you omit that option, the messages are browsed in physical order. If you specify MQGMO_BROWSE_FIRST, you can switch between logical order and physical order. Subsequent MQGET calls using MQGMO_BROWSE_NEXT browse the queue in the same order as the most recent call that specified MQGMO_BROWSE_FIRST for the queue handle. The call fails with reason code MQRC_INCONSISTENT_BROWSE if this condition is not satisfied.

Note: Take special care when using an MQGET call to browse beyond the end of a message group if MQGMO_LOGICAL_ORDER is not specified. For example, suppose the last message in the group precedes the first message in the group on the queue. Using MQGMO_BROWSE_NEXT to browse beyond the end of the group, specifying MQMO_MATCH_MSG_SEQ_NUMBER with *MsgSeqNumber* set to 1 returns the first message in the group already browsed. This result can happen immediately, or a number of MQGET calls later if there are intervening groups. The same consideration applies for a logical message not in a group.
The group and segment information for browse calls are retained separately from the information for calls that remove messages from the queue.

MQGMO_BROWSE_MSG_UNDER_CURSOR
Retrieve the message pointed to by the browse cursor nondestructively, regardless of the MQMO_* options specified in the *MatchOptions* field in MQGMO.

MQGMO_BROWSE_MSG_UNDER_CURSOR is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

It is also an error if the queue was not opened for browse.
The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option. The call fails if neither of these calls has been issued for this queue since it was opened. The call also fails if the message that was under the browse cursor has since been retrieved destructively.
The position of the browse cursor is not changed by this call.
The MQGMO_MSG_UNDER_CURSOR option can be used with a non-browse MQGET call, to remove the message from the queue.
The browse cursor is not moved by a non-browse MQGET call, even if using the same *Hobj* handle. Nor is it moved by a browse MQGET call that returns a completion code of MQCC_FAILED, or a reason code of MQRC_TRUNCATED_MSG_FAILED.
If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified with MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor, so that is returned without unlocking and locking again. The message remains locked.
- If there is no locked message and there is a message under the browse cursor, it is locked and returned to the application. If there is no message under the browse cursor, the call fails.

If MQGMO_BROWSE_MSG_UNDER_CURSOR is specified without MQGMO_LOCK:

- If there is already a message locked, it must be the one under the cursor. The message is returned to the application and then unlocked. Because the message is now unlocked, there is no guarantee that it can be browsed again, or retrieved destructively by the same application. It might have been retrieved destructively by another application getting messages from the queue.
- If there is no locked message and there is a message under the browse cursor, it is returned to the application. If there is no message under the browse cursor the call fails.

If MQGMO_COMPLETE_MSG is specified with MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must identify a message whose *Offset* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.
The group and segment information for browse calls are retained separately from the information for calls that remove messages from the queue.

MQGMO_MSG_UNDER_CURSOR
Retrieve the message pointed to by the browse cursor, regardless of the MQMO_* options specified in the *MatchOptions* field in MQGMO. The message is removed from the queue.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR, the browse cursor must identify a message whose *Offset* field in MQMD is zero. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

This option is not valid with any of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT
- MQGMO_UNLOCK

It is also an error if the queue was not opened both for browse and for input. If the browse cursor is not currently pointing to a retrievable message, an error is returned by the MQGET call.

MQGMO_MARK_BROWSE_HANDLE
The message that is returned by a successful MQGET, or identified by the returned *MsgToken*, is marked. The mark is specific to the object handle used in the call.

The message is not removed from the queue.

MQGMO_MARK_BROWSE_HANDLE is valid only if one of the following options is also specified:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT

MQGMO_MARK_BROWSE_HANDLE is not valid with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_UNLOCK

The message remains in this state until one of the following events occurs:

- The object handle concerned is closed, either normally or otherwise.
- The message is unmarked for this handle by a call to MQGET with the option MQGMO_UNMARK_BROWSE_HANDLE.
- The message is returned from a call to destructive MQGET, which completes with MQCC_OK or MQCC_WARNING. The message state remains changed even if the MQGET is later rolled-back.
- The message expires.

MQGMO_MARK_BROWSE_CO_OP
The message that is returned by a successful MQGET, or identified by the returned *MsgToken*, is marked for all handles in the cooperating set.

The cooperative level mark is in addition to any handle level mark that might have been set.

The message is not removed from the queue.

MQGMO_MARK_BROWSE_CO_OP is valid only if the object handle used was returned by a call to MQOPEN that specified MQOO_CO_OP. You must also specify one of the following MQGMO options:

- MQGMO_BROWSE_FIRST

- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_BROWSE_NEXT

This option is not valid with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_UNLOCK

If the message is already marked, and the option MQGMO_UNMARKED_BROWSE_MSG is not specified, the call fails with MQCC_FAILED and reason code MQRC_MSG_MARKED_BROWSE_CO_OP.

The message remains in this state until one of the following events occurs:

- All object handles in the cooperating set are closed.
- The message is unmarked for cooperating browsers by a call to MQGET with the option MQGMO_UNMARK_BROWSE_CO_OP.
- The message is automatically unmarked by the queue manager.
- The message is returned from a call to a non-browse MQGET. The message state remains changed even if the MQGET is later rolled-back.
- The message expires.

MQGMO_UNMARKED_BROWSE_MSG

A call to MQGET that specifies MQGMO_UNMARKED_BROWSE_MSG returns a message that is considered to be unmarked for its handle. It does not return a message if the message was marked for its handle. It also does not return the message if the queue was opened by a call to MQOPEN, with the option MQOO_CO_OP, and the message has been marked by a member of the cooperating set.

This option is not valid with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_UNLOCK

MQGMO_UNMARK_BROWSE_CO_OP

After a call to MQGET that specifies this option, the message is no longer considered by any open handles in the set of cooperating handles to be marked for the cooperating set. The message is still considered to be marked at handle level if it was marked at handle level before this call.

Using MQGMO_UNMARK_BROWSE_CO_OP is valid only with a handle returned by a successful call to MQOPEN with the option MQOO_CO_OP. The MQGET succeeds even if the message is not considered to be marked by the cooperating set of handles.

MQGMO_UNMARK_BROWSE_CO_OP is not valid on a non-browse MQGET call, or with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_MARK_BROWSE_CO_OP

- MQGMO_UNLOCK
- MQGMO_UNMARKED_BROWSE_MSG

MQGMO_UNMARK_BROWSE_HANDLE
After a call to MQGET that specifies this option, the message located is no longer considered to be marked by this handle.

The call succeeds even if the message is not marked for this handle.

This option is not valid on a non-browse MQGET call, or with any of the following options:

- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_COMPLETE_MSG
- MQGMO_LOCK
- MQGMO_LOGICAL_ORDER
- MQGMO_MARK_BROWSE_CO_OP
- MQGMO_UNLOCK
- MQGMO_UNMARKED_BROWSE_MSG

Lock options: The following options relate to locking messages on the queue:

MQGMO_LOCK
Lock the message that is browsed, so that the message becomes invisible to any other handle open for the queue. The option can be specified only if one of the following options is also specified:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR

Only one message can be locked for each queue handle. The message can be a logical message or a physical message:

- If you specify MQGMO_COMPLETE_MSG, all the message segments that make up the logical message are locked to the queue handle. The messages must all be present on the queue and available for retrieval.
- If you omit MQGMO_COMPLETE_MSG, only a single physical message is locked to the queue handle. If this message happens to be a segment of a logical message, the locked segment prevents other applications using MQGMO_COMPLETE_MSG to retrieve or browse the logical message.

The locked message is always the one under the browse cursor. The message can be removed from the queue by a later MQGET call that specifies the MQGMO_MSG_UNDER_CURSOR option. Other MQGET calls using the queue handle can also remove the message (for example, a call that specifies the message identifier of the locked message).

If the call returns completion code MQCC_FAILED, or MQCC_WARNING with reason code MQRC_TRUNCATED_MSG_FAILED, no message is locked.

If the application does not remove the message from the queue, the lock is released by one of the following actions:

- Issuing another MQGET call for this handle, specifying either MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT. The lock is released if the call completes with MQCC_OK or MQCC_WARNING. The message remains locked if the call completes with MQCC_FAILED. However, the following exceptions apply:
  - The message is not unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
  - The message is unlocked if MQCC_FAILED is returned with MQRC_NO_MSG_AVAILABLE.

If you also specify MQGMO_LOCK, the message returned is locked. If you omit MQGMO_LOCK, there is no locked message after the call.

If you specify MQGMO_WAIT, and no message is immediately available, the original message is unlocked before the start of the wait.

- Issuing another MQGET call for this handle, with MQGMO_BROWSE_MSG_UNDER_CURSOR, without MQGMO_LOCK. The lock is released if the call completes with MQCC_OK or MQCC_WARNING. The message remains locked if the call completes with MQCC_FAILED. However, the following exception applies:
  - The message is not unlocked if MQCC_WARNING is returned with MQRC_TRUNCATED_MSG_FAILED.
- Issuing another MQGET call for this handle with MQGMO_UNLOCK.
- Issuing an MQCLOSE call using the handle. The MQCLOSE might be implicit, caused by the application ending.

No special MQOPEN option is required to specify MQGMO_LOCK, other than MQOO_BROWSE, which is needed to specify an accompanying browse option.

MQGMO_LOCK is not valid with any of the following options:

- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_UNLOCK

MQGMO_LOCK is not possible when you are using a IBM WebSphere MQ client on HP Integrity NonStop Server to a z/OS queue manager when coordinated by TMF.

MQGMO_UNLOCK

The message to be unlocked must have been previously locked by an MQGET call with the MQGMO_LOCK option. If there is no message locked for this handle, the call completes with MQCC_WARNING and MQRC_NO_MSG_LOCKED.

The *MsgDesc*, *BufferLength*, *Buffer*, and *DataLength* parameters are not checked or altered if you specify MQGMO_UNLOCK. No message is returned in *Buffer*.

No special open option is required to specify MQGMO_UNLOCK (although MQOO_BROWSE is needed to issue the lock request in the first place).

This option is not valid with any options except the following:

- MQGMO_NO_WAIT
- MQGMO_NO_SYNCPOINT

Both of these options are assumed whether specified or not.

Message-data options: The following options relate to the processing of the message data when the message is read from the queue:

MQGMO_ACCEPT_TRUNCATED_MSG

If the message buffer is too small to hold the complete message, allow the MQGET call to fill the buffer. MQGET fills the buffer with as much of the message it can. It issues a warning completion code, and completes its processing. This means that:

- When browsing messages, the browse cursor is advanced to the returned message.
- When removing messages, the returned message is removed from the queue.
- Reason code MQRC_TRUNCATED_MSG_ACCEPTED is returned if no other error occurs.

Without this option, the buffer is still filled with as much of the message as it can hold. A warning completion code is issued, but processing is not completed. This means that:

- When browsing messages, the browse cursor is not advanced.
- When removing messages, the message is not removed from the queue.

- Reason code MQRC_TRUNCATED_MSG_FAILED is returned if no other error occurs.

MQGMO_CONVERT

This option converts the application data in the message to conform to the *CodedCharSetId* and *Encoding* values specified in the *MsgDesc* parameter on the MQGET call. The data is converted before it is copied to the *Buffer* parameter.

The *Format* field specified when the message was put is assumed by the conversion process to identify the nature of the data in the message. The message data is converted by the queue manager for built-in formats, and by a user-written exit for other formats. See "Data conversion" on page 2210 for details of the data-conversion exit.

- If conversion is successful, the *CodedCharSetId* and *Encoding* fields specified in the *MsgDesc* parameter are unchanged on return from the MQGET call.
- If only conversion fails the message data is returned unconverted The *CodedCharSetId* and *Encoding* fields in *MsgDesc* are set to the values for the unconverted message. The completion code is MQCC_WARNING in this case.

In either case, these fields describe the character-set identifier and encoding of the message data that is returned in the *Buffer* parameter.

See the *Format* field described in "MQMD - Message descriptor" on page 1705 for a list of format names for which the queue manager performs the conversion.

Group and segment options: The following options relate to the processing of messages in groups and segments of logical messages. Before the option descriptions, here are some definitions of important terms:

Physical message

A physical message is the smallest unit of information that can be placed on or removed from a queue. It often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor, MQMD. Typically, physical messages are distinguished by differing values for the message identifier, the *MsgId* field in MQMD. The queue manager does not enforce different values.

Logical message

A logical message is a single unit of application information. In the absence of system constraints, a logical message is the same as a physical message. If logical messages are large, system constraints might make it advisable or necessary to split a logical message into two or more physical messages, called segments.

A logical message that has been segmented consists of two or more physical messages that have the same nonnull group identifier, *GroupId* field in MQMD. They have the same message sequence number, *MsgSeqNumber* field in MQMD. The segments are distinguished by differing values for the segment offset, *Offset* field in MQMD. The segment offset is the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message typically have different message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a nonnull group identifier. In this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages, for which segmentation has been inhibited by the sending application, have a null group identifier, MQGI_NONE, unless the logical message belongs to a message group.

Message group

A message group is a set of one or more logical messages that have the same nonnull group identifier. The logical messages in the group are distinguished by different values for the message sequence number. The sequence number is an integer in the range 1 through n, where n is the number of logical messages in the group. If one or more of the logical messages is segmented, there are more than n physical messages in the group.

MQGMO_LOGICAL_ORDER

MQGMO_LOGICAL_ORDER controls the order in which messages are returned by successive MQGET calls for the queue handle. The option must be specified on each call.

If MQGMO_LOGICAL_ORDER is specified for successive MQGET calls for the same queue handle, messages in groups are returned in the order of their message sequence numbers. Segments of logical messages are returned in the order given by their segment offsets. This order might be different from the order in which those messages and segments occur on the queue.

Note: Specifying MQGMO_LOGICAL_ORDER has no adverse consequences on messages that do not belong to groups and that are not segments. In effect, such messages are treated as though each belonged to a message group consisting of only one message. It is safe to specify MQGMO_LOGICAL_ORDER when retrieving messages from queues that contain a mixture of messages in groups, message segments, and unsegmented messages not in groups.

To return the messages in the required order, the queue manager retains the group and segment information between successive MQGET calls. The group and segment information identifies the current message group and current logical message for the queue handle. It also identifies the current position within the group and logical message, and whether the messages are being retrieved within a unit of work. Because the queue manager retains this information, the application does not need to set the group and segment information before each MQGET call. Specifically, it means that the application does not need to set the *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD. However, the application must set the MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT option correctly on each call.

When the queue is opened, there is no current message group and no current logical message. A message group becomes the current message group when a message that has the MQMF_MSG_IN_GROUP flag is returned by the MQGET call. With MQGMO_LOGICAL_ORDER specified on successive calls, that group remains the current group until a message is returned that has:

- MQMF_LAST_MSG_IN_GROUP without MQMF_SEGMENT (that is, the last logical message in the group is not segmented), or
- MQMF_LAST_MSG_IN_GROUP with MQMF_LAST_SEGMENT (that is, the message returned is the last segment of the last logical message in the group).

When such a message is returned, the message group is terminated, and on successful completion of the MQGET call there is no longer a current group. In a similar way, a logical message becomes the current logical message when a message that has the MQMF_SEGMENT flag is returned by the MQGET call. The logical message is terminated when the message that has the MQMF_LAST_SEGMENT flag is returned.

If no selection criteria are specified, successive MQGET calls return, in the correct order, the messages for the first message group on the queue. They then return the messages for the second message group, and so on, until there are no more messages available. It is possible to select the particular message groups returned by specifying one or more of the following options in the *MatchOptions* field:

- MQMO_MATCH_MSG_ID
- MQMO_MATCH_CORREL_ID
- MQMO_MATCH_GROUP_ID

However, these options are effective only when there is no current message group or logical message. See the *MatchOptions* field described in "MQGMO - Get-message options" on page 1655 for further details.

Table 154 on page 1675 shows the values of the *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* fields that the queue manager looks for when attempting to find a message to return on the MQGET call. The rules apply both to removing messages from the queue, and browsing messages on the queue. In the table, Either means Yes or No:

LOG ORD
Indicates whether the MQGMO_LOGICAL_ORDER option is specified on the call.

Cur grp
Indicates whether a current message group exists before the call.

Cur log msg
Indicates whether a current logical message exists before the call.

Other columns
Show the values that the queue manager looks for. Previous denotes the value returned for the field in the previous message for the queue handle.

Table 154. MQGET options relating to messages in groups and segments of logical messages

Options you specify	Group and log-msg status before call		Values the queue manager looks for				
LOG ORD	**Cur grp**	**Cur log msg**	*MsgId*	*CorrelId*	*GroupId*	*MsgSeqNumber*	*Offset*
Yes	No	No	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	1	0
Yes	No	Yes	Any message identifier	Any correlation identifier	Previous group identifier	1	Previous offset + previous segment length
Yes	Yes	No	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number + 1	0
Yes	Yes	Yes	Any message identifier	Any correlation identifier	Previous group identifier	Previous sequence number	Previous offset + previous segment length
No	Either	Either	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*	Controlled by *MatchOptions*

If multiple message groups are present on the queue and eligible for return, the groups are returned in the order determined by the position on the queue of the first segment of the first logical message in each group. That is, the physical messages that have message sequence numbers of 1, and offsets of 0, determine the order in which eligible groups are returned.

The MQGMO_LOGICAL_ORDER option affects units of work as follows:

- If the first logical message or segment in a group is retrieved within a unit of work, all the other logical messages and segments in the group must be retrieved within a unit of work, if the same queue handle is used. However, they need not be retrieved within the same unit of work. This allows a message group consisting of many physical messages to be split across two or more consecutive units of work for the queue handle.
- If the first logical message or segment in a group is *not* retrieved within a unit of work, and the same queue handle is used, none of the other logical messages and segments in the group can be retrieved within a unit of work.

If these conditions are not satisfied, the MQGET call fails with reason code MQRC_INCONSISTENT_UOW.

When MQGMO_LOGICAL_ORDER is specified, the MQGMO supplied on the MQGET call must not be less than MQGMO_VERSION_2, and the MQMD must not be less than MQMD_VERSION_2. If this condition is not satisfied, the call fails with reason code MQRC_WRONG_GMO_VERSION or MQRC_WRONG_MD_VERSION, as appropriate.

If MQGMO_LOGICAL_ORDER is *not* specified for successive MQGET calls for the queue handle, messages are returned without regard for whether they belong to message groups, or whether

they are segments of logical messages. This means that messages or segments from a particular group or logical message might be returned out of order, or intermingled with messages or segments from other groups or logical messages, or with messages that are not in groups and are not segments. In this situation, the particular messages that are returned by successive MQGET calls is controlled by the MQMO_* options specified on those calls (see the *MatchOptions* field described in "MQGMO - Get-message options" on page 1655 for details of these options).

This is the technique that can be used to restart a message group or logical message in the middle, after a system failure has occurred. When the system restarts, the application can set the *GroupId*, *MsgSeqNumber*, *Offset*, and *MatchOptions* fields to the appropriate values, and then issue the MQGET call with MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT set, but *without* specifying MQGMO_LOGICAL_ORDER. If this call is successful, the queue manager retains the group and segment information, and subsequent MQGET calls using that queue handle can specify MQGMO_LOGICAL_ORDER as normal.

The group and segment information that the queue manager retains for the MQGET call is separate from the group and segment information that it retains for the MQPUT call. In addition, the queue manager retains separate information for:

- MQGET calls that remove messages from the queue.
- MQGET calls that browse messages on the queue.

For any given queue handle, the application can mix MQGET calls that specify MQGMO_LOGICAL_ORDER with MQGET calls that do not. However, note the following points:

- If you omit MQGMO_LOGICAL_ORDER, each successful MQGET call causes the queue manager to set the saved group and segment information to the values corresponding to the message returned; this replaces the existing group and segment information retained by the queue manager for the queue handle. Only the information appropriate to the action of the call (browse or remove) is modified.
- If you omit MQGMO_LOGICAL_ORDER, the call does not fail if there is a current message group or logical message; the call might succeed with an MQCC_WARNING completion code. Table 155 shows the various cases that can arise. In these cases, if the completion code is not MQCC_OK, the reason code is one of the following (as appropriate):
  - MQRC_INCOMPLETE_GROUP
  - MQRC_INCOMPLETE_MSG
  - MQRC_INCONSISTENT_UOW

  Note: The queue manager does not check the group and segment information when browsing a queue, or when closing a queue that was opened for browse but not input; in those cases the completion code is always MQCC_OK (assuming no other errors).

Table 155. Outcome when MQGET or MQCLOSE call is not consistent with group and segment information

Current call is	Previous call was MQGET with MQGMO_LOGICAL_ORDER	Previous call was MQGET without MQGMO_LOGICAL_ORDER
MQGET with MQGMO_LOGICAL_ORDER	MQCC_FAILED	MQCC_FAILED
MQGET without MQGMO_LOGICAL_ORDER	MQCC_WARNING	MQCC_OK
MQCLOSE with an unterminated group or logical message	MQCC_WARNING	MQCC_OK

Applications that want to retrieve messages and segments in logical order are recommended to specify MQGMO_LOGICAL_ORDER, as this is the simplest option to use. This option relieves the application of the need to manage the group and segment information, because the queue manager manages that information. However, specialized applications might need more control than that provided by the MQGMO_LOGICAL_ORDER option, and this can be achieved by not

specifying that option. The application must then ensure that the *MsgId*, *CorrelId*, *GroupId*, *MsgSeqNumber*, and *Offset* fields in MQMD, and the MQMO_* options in *MatchOptions* in MQGMO, are set correctly, before each MQGET call.

For example, an application that wants to *forward* physical messages that it receives, without regard for whether those messages are in groups or segments of logical messages, must *not* specify MQGMO_LOGICAL_ORDER. In a complex network with multiple paths between sending and receiving queue managers, the physical messages might arrive out of order. By specifying neither MQGMO_LOGICAL_ORDER, nor the corresponding MQPMO_LOGICAL_ORDER on the MQPUT call, the forwarding application can retrieve and forward each physical message as soon as it arrives, without having to wait for the next one in logical order to arrive.

You can specify MQGMO_LOGICAL_ORDER with any of the other MQGMO_* options, and with various of the MQMO_* options in appropriate circumstances (see above).

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue maps to must be at CFLEVEL(3) or CFLEVEL(4).
- On AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems, this option is supported for all local queues.

MQGMO_COMPLETE_MSG

Only a complete logical message can be returned by the MQGET call. If the logical message is segmented, the queue manager reassembles the segments and returns the complete logical message to the application; the fact that the logical message was segmented is not apparent to the application retrieving it.

Note: This is the only option that causes the queue manager to reassemble message segments. If not specified, segments are returned individually to the application if they are present on the queue (and they satisfy the other selection criteria specified on the MQGET call). Applications that do not want to receive individual segments must always specify MQGMO_COMPLETE_MSG.

To use this option, the application must provide a buffer that is big enough to accommodate the complete message, or specify the MQGMO_ACCEPT_TRUNCATED_MSG option.

If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_COMPLETE_MSG prevents the retrieval of segments belonging to incomplete logical messages. However, those message segments still contribute to the value of the *CurrentQDepth* queue attribute; this means that there might be no retrievable logical messages, even though *CurrentQDepth* is greater than zero.

For *persistent* messages, the queue manager can reassemble the segments only within a unit of work:

- If the MQGET call is operating within a user-defined unit of work, that unit of work is used. If the call fails during the reassembly process, the queue manager reinstates on the queue any segments that were removed during reassembly. However, the failure does not prevent the unit of work being committed successfully.
- If the call is operating outside a user-defined unit of work, and there is no user-defined unit of work in existence, the queue manager creates a unit of work for the duration of the call. If the call is successful, the queue manager commits the unit of work automatically (the application does not need to do this). If the call fails, the queue manager backs out the unit of work.
- If the call is operating outside a user-defined unit of work, but a user-defined unit of work exists, the queue manager cannot reassemble. If the message does not require reassembly, the call can still succeed. But if the message requires reassembly, the call fails with reason code MQRC_UOW_NOT_AVAILABLE.

For *nonpersistent* messages, the queue manager does not require a unit of work to be available to perform reassembly.

Each physical message that is a segment has its own message descriptor. For the segments constituting a single logical message, most of the fields in the message descriptor are the same for all segments in the logical message; typically it is only the *MsgId*, *Offset*, and *MsgFlags* fields that differ between segments in the logical message. However, if a segment is placed on a dead-letter queue at an intermediate queue manager, the DLQ handler retrieves the message specifying the MQGMO_CONVERT option, and this can result in the character set or encoding of the segment being changed. If the DLQ handler successfully sends the segment on its way, the segment might have a character set or encoding that differs from the other segments in the logical message when the segment arrives at the destination queue manager.

A logical message consisting of segments in which the *CodedCharSetId* and *Encoding* fields differ cannot be reassembled by the queue manager into a single logical message. Instead, the queue manager reassembles and returns the first few consecutive segments at the start of the logical message that have the same character-set identifiers and encodings, and the MQGET call completes with completion code MQCC_WARNING and reason code MQRC_INCONSISTENT_CCSIDS or MQRC_INCONSISTENT_ENCODINGS, as appropriate. This happens regardless of whether MQGMO_CONVERT is specified. To retrieve the remaining segments, the application must reissue the MQGET call without the MQGMO_COMPLETE_MSG option, retrieving the segments one by one. MQGMO_LOGICAL_ORDER can be used to retrieve the remaining segments in order.

An application that puts segments can also set other fields in the message descriptor to values that differ between segments. However, there is no advantage in doing this if the receiving application uses MQGMO_COMPLETE_MSG to retrieve the logical message. When the queue manager reassembles a logical message, it returns in the message descriptor the values from the message descriptor for the *first* segment; the only exception is the *MsgFlags* field, which the queue manager sets to indicate that the reassembled message is the only segment.

If MQGMO_COMPLETE_MSG is specified for a report message, the queue manager performs special processing. The queue manager checks the queue to see if all the report messages of that report type relating to the different segments in the logical message are present on the queue. If they are, they can be retrieved as a single message by specifying MQGMO_COMPLETE_MSG. For this to be possible, either the report messages must be generated by a queue manager or MCA which supports segmentation, or the originating application must request at least 100 bytes of message data (that is, the appropriate MQRO_*_WITH_DATA or MQRO_*_WITH_FULL_DATA options must be specified). If less than the full amount of application data is present for a segment, the missing bytes are replaced by nulls in the report message returned.

If MQGMO_COMPLETE_MSG is specified with MQGMO_MSG_UNDER_CURSOR or MQGMO_BROWSE_MSG_UNDER_CURSOR, the browse cursor must be positioned on a message whose *Offset* field in MQMD has a value of 0. If this condition is not satisfied, the call fails with reason code MQRC_INVALID_MSG_UNDER_CURSOR.

MQGMO_COMPLETE_MSG implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_COMPLETE_MSG can be specified with any of the other MQGMO_* options apart from MQGMO_SYNCPOINT_IF_PERSISTENT, and with any of the MQMO_* options apart from MQMO_MATCH_OFFSET.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3) or CFLEVEL(4).
- On AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems, this option is supported for all local queues.

MQGMO_ALL_MSGS_AVAILABLE

Messages in a group become available for retrieval only when *all* messages in the group are available. If the queue contains message groups with some of the messages missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_MSGS_AVAILABLE prevents retrieval of messages belonging to incomplete groups.

However, those messages still contribute to the value of the *CurrentQDepth* queue attribute; this means that there may be no retrievable message groups, even though *CurrentQDepth* is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_MSGS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_MSGS_AVAILABLE has an effect *only* when there is no current group or logical message. If there *is* a current group or logical message, MQGMO_ALL_MSGS_AVAILABLE is ignored. This means that MQGMO_ALL_MSGS_AVAILABLE can remain on when processing messages in logical order.
- If MQGMO_ALL_MSGS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_MSGS_AVAILABLE *always* has an effect. This means that the option must be turned off after the first message in the group has been removed from the queue, in order to be able to remove the remaining messages in the group.

Successful completion of an MQGET call specifying MQGMO_ALL_MSGS_AVAILABLE means that at the time that the MQGET call was issued, all the messages in the group were on the queue. However, be aware that other applications can still remove messages from the group (the group is not locked to the application that retrieves the first message in the group).

If you omit this option, messages belonging to groups can be retrieved even when the group is incomplete.

MQGMO_ALL_MSGS_AVAILABLE implies MQGMO_ALL_SEGMENTS_AVAILABLE, which need not therefore be specified.

MQGMO_ALL_MSGS_AVAILABLE can be specified with any of the other MQGMO_* options, and with any of the MQMO_* options.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3) or CFLEVEL(4).
- On AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems, this option is supported for all local queues.

MQGMO_ALL_SEGMENTS_AVAILABLE

Segments in a logical message become available for retrieval only when *all* segments in the logical message are available. If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_SEGMENTS_AVAILABLE prevents retrieval of segments belonging to incomplete logical messages. However, those segments still contribute to the value of the *CurrentQDepth* queue attribute; this means that there might be no retrievable logical messages, even though *CurrentQDepth* is greater than zero. If there are no other messages that are retrievable, reason code MQRC_NO_MSG_AVAILABLE is returned after the specified wait interval (if any) has expired.

The processing of MQGMO_ALL_SEGMENTS_AVAILABLE depends on whether MQGMO_LOGICAL_ORDER is also specified:

- If both options are specified, MQGMO_ALL_SEGMENTS_AVAILABLE has an effect *only* when there is no current logical message. If there *is* a current logical message, MQGMO_ALL_SEGMENTS_AVAILABLE is ignored. This means that MQGMO_ALL_SEGMENTS_AVAILABLE can remain on when processing messages in logical order.
- If MQGMO_ALL_SEGMENTS_AVAILABLE is specified without MQGMO_LOGICAL_ORDER, MQGMO_ALL_SEGMENTS_AVAILABLE *always* has an effect. This means that the option must be turned off after the first segment in the logical message has been removed from the queue, in order to be able to remove the remaining segments in the logical message.

If this option is not specified, message segments can be retrieved even when the logical message is incomplete.

While both MQGMO_COMPLETE_MSG and MQGMO_ALL_SEGMENTS_AVAILABLE require all segments to be available before any of them can be retrieved, the former returns the complete message, whereas the latter allows the segments to be retrieved one by one.

If MQGMO_ALL_SEGMENTS_AVAILABLE is specified for a report message, the queue manager checks the queue to see if there is at least one report message for each of the segments that make up the complete logical message. If there is, the MQGMO_ALL_SEGMENTS_AVAILABLE condition is satisfied. However, the queue manager does not check the *type* of the report messages present, and so there might be a mixture of report types in the report messages relating to the segments of the logical message. As a result, the success of MQGMO_ALL_SEGMENTS_AVAILABLE does not imply that MQGMO_COMPLETE_MSG will succeed. If there *is* a mixture of report types present for the segments of a particular logical message, those report messages must be retrieved one by one.

You can specify MQGMO_ALL_SEGMENTS_AVAILABLE with any of the other MQGMO_* options, and with any of the MQMO_* options.

- On z/OS, this option is supported for private and shared queues, but the queue must have an index type of MQIT_GROUP_ID. For shared queues, the CFSTRUCT object that the queue map to must be at CFLEVEL(3) or CFLEVEL(4).
- On AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems, this option is supported for all local queues.

Property options: The following options relate to the properties of the message:

MQGMO_PROPERTIES_AS_Q_DEF

Properties of the message, except those contained in the message descriptor (or extension) should be represented as defined by the *PropertyControl* queue attribute. If a *MsgHandle* is provided this option is ignored and the properties of the message are available via the *MsgHandle*, unless the value of the *PropertyControl* queue attribute is MQPROP_FORCE_MQRFH2.

This is the default action if no property options are specified.

MQGMO_PROPERTIES_IN_HANDLE

Properties of the message should be made available via the *MsgHandle*. If no message handle is provided the call fails with reason MQRC_HMSG_ERROR.

Note: If the message is later read by an application that does not create a message handle, the queue manager places any message properties into an MQRFH2 structure. You might find that the presence of an unexpected MQRFH2 header disrupts the behavior of an existing application.

MQGMO_NO_PROPERTIES

No properties of the message, except those contained in the message descriptor (or extension) will be retrieved. If a *MsgHandle* is provided it will be ignored.

MQGMO_PROPERTIES_FORCE_MQRFH2

Properties of the message, except those contained in the message descriptor (or extension) should be represented using MQRFH2 headers. This provides compatibility with earlier version for applications which are expecting to retrieve properties but are unable to be changed to use message handles. If a *MsgHandle* is provided it is ignored.

MQGMO_PROPERTIES_COMPATIBILITY

If the message contains a property with a prefix of **"mcd."**, **"jms."**, **"usr."**, or **"mqext."**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

Default option: If none of the options described is required, the following option can be used:

MQGMO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values. MQGMO_NONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of the *Options* field is MQGMO_NO_WAIT plus MQGMO_PROPERTIES_AS_Q_DEF.

Reserved1 (MQCHAR):

This is a reserved field. The initial value of this field is a blank character. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Reserved2 (MQLONG):

This is a reserved field. The initial value of this field is a blank character. This field is ignored if *Version* is less than **MQGMO_VERSION_4**.

ResolvedQName (MQCHAR48):

This is an output field that the queue manager sets to the local name of the queue from which the message was retrieved, as defined to the local queue manager. This is different from the name used to open the queue if:

- An alias queue was opened (in which case, the name of the local queue to which the alias resolved is returned), or
- A model queue was opened (in which case, the name of the dynamic local queue is returned).

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ReturnedLength (MQLONG):

This is an output field that the queue manager sets to the length in bytes of the message data returned by the MQGET call in the *Buffer* parameter. If the queue manager does not support this capability, *ReturnedLength* is set to the value MQRL_UNDEFINED.

When messages are converted between encodings or character sets, the message data can sometimes change size. On return from the MQGET call:

- If *ReturnedLength* is *not* MQRL_UNDEFINED, the number of bytes of message data returned is given by *ReturnedLength*.
- If *ReturnedLength* has the value MQRL_UNDEFINED, the number of bytes of message data returned is usually given by the smaller of *BufferLength* and *DataLength*, but can be *less than* this if the MQGET call completes with reason code MQRC_TRUNCATED_MSG_ACCEPTED. If this happens, the insignificant bytes in the *Buffer* parameter are set to nulls.

The following special value is defined:

MQRL_UNDEFINED
Length of returned data not defined.

On z/OS, the value returned for the *ReturnedLength* field is always MQRL_UNDEFINED.

The initial value of this field is MQRL_UNDEFINED. This field is ignored if *Version* is less than MQGMO_VERSION_3.

Segmentation (MQCHAR):

This is a flag that indicates whether further segmentation is allowed for the message retrieved. It has one of the following values:

MQSEG_INHIBITED
Segmentation not allowed.

MQSEG_ALLOWED
Segmentation allowed.

On z/OS, the queue manager always sets this field to MQSEG_INHIBITED.

This is an output field. The initial value of this field is MQSEG_INHIBITED. This field is ignored if *Version* is less than MQGMO_VERSION_2.

SegmentStatus (MQCHAR):

This is a flag that indicates whether the message retrieved is a segment of a logical message. It has one of the following values:

MQSS_NOT_A_SEGMENT
Message is not a segment.

MQSS_SEGMENT
Message is a segment, but is not the last segment of the logical message.

MQSS_LAST_SEGMENT
Message is the last segment of the logical message.

This is also the value returned if the logical message consists of only one segment.

On z/OS, the queue manager always sets this field to MQSS_NOT_A_SEGMENT.

This is an output field. The initial value of this field is MQSS_NOT_A_SEGMENT. This field is ignored if *Version* is less than MQGMO_VERSION_2.

Signal1 (MQLONG):

This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option; it identifies a signal that is to be delivered when a message is available.

Note: The data type and usage of this field are determined by the environment; for this reason, applications that you want to port between different environments must not use signals.

- On z/OS, this field must contain the address of an Event Control Block (ECB). The ECB must be cleared by the application before the MQGET call is issued. The storage containing the ECB must not be freed until the queue is closed. The ECB is posted by the queue manager with one of the signal completion codes described. These completion codes are set in bits 2 through 31 of the ECB, the area defined in the z/OS mapping macro IHAECB as being for a user completion code.
- In all other environments, this is a reserved field; its value is not significant.

The signal completion codes are:

MQEC_MSG_ARRIVED
A suitable message has arrived on the queue. This message has not been reserved for the caller; a second MQGET request must be issued, but another application might retrieve the message before the second request is made.

MQEC_WAIT_INTERVAL_EXPIRED
The specified *WaitInterval* has expired without a suitable message arriving.

MQEC_WAIT_CANCELED
The wait was canceled for an indeterminate reason (such as the queue manager terminating or the queue being disabled). Reissue the request if you want further diagnosis.

MQEC_Q_MGR_QUIESCING
The wait was canceled because the queue manager has entered the quiescing state (MQGMO_FAIL_IF_QUIESCING was specified on the MQGET call).

MQEC_CONNECTION_QUIESCING
The wait was canceled because the connection has entered the quiescing state (MQGMO_FAIL_IF_QUIESCING was specified on the MQGET call).

The initial value of this field is determined by the environment:

- On z/OS, the initial value is the null pointer.
- In all other environments, the initial value is `0`.

Signal2 (MQLONG):

This is an input field that is used only in conjunction with the MQGMO_SET_SIGNAL option. It is a reserved field; its value is not significant.

The initial value of this field is `0`.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQGMO_STRUC_ID
Identifier for get-message options structure.

For the C programming language, the constant MQGMO_STRUC_ID_ARRAY is also defined; this has the same value as MQGMO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQGMO_STRUC_ID.

Version (MQLONG):

Version is the structure version number.

The value must be one of the following:

MQGMO_VERSION_1
Version-1 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_2
Version-2 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_3
Version-3 get-message options structure.

This version is supported in all environments.

MQGMO_VERSION_4
Version-4 get-message options structure.

This version is supported in all environments.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQGMO_CURRENT_VERSION
Current version of get-message options structure.

This is always an input field. The initial value of this field is MQGMO_VERSION_1.

WaitInterval (MQLONG):

This is the approximate time, expressed in milliseconds, that the MQGET call waits for a suitable message to arrive (that is, a message satisfying the selection criteria specified in the *MsgDesc* parameter of the MQGET call.

.

Important: There is no wait, or delay, if a suitable message is available immediately.

See the *MsgId* field described in "MQMD - Message descriptor" on page 1705 for more details). If no suitable message has arrived after this time has elapsed, the call completes with MQCC_FAILED and reason code MQRC_NO_MSG_AVAILABLE.

On z/OS, the period of time that the MQGET call actually waits is affected by system loading and work-scheduling considerations, and can vary between the value specified for *WaitInterval* and approximately 250 milliseconds greater than *WaitInterval*.

WaitInterval is used in conjunction with the MQGMO_WAIT or MQGMO_SET_SIGNAL option. It is ignored if neither of these is specified. If one of these is specified, *WaitInterval* must be greater than or equal to zero, or the following special value:

MQWI_UNLIMITED
Unlimited wait interval.

The initial value of this field is `0`.

Initial values and language declarations for MQGMO:

Table 156. Initial values of fields in MQGMO for MQGMO

Field name	Name of constant	Value of constant
StrucId	MQGMO_STRUC_ID	`'GMO¬'`
Version	MQGMO_VERSION_1	`1`
Options	MQGMO_NO_WAIT	`0`
WaitInterval	None	`0`
Signal1	None	Null pointer on z/OS; `0` otherwise
Signal2	None	`0`
ResolvedQName	None	Null string or blanks
MatchOptions	MQMO_MATCH_MSG_ID + MQMO_MATCH_CORREL_ID	`3`
GroupStatus	MQGS_NOT_IN_GROUP	`'⌂'`
SegmentStatus	MQSS_NOT_A_SEGMENT	`'⌂'`
Segmentation	MQSEG_INHIBITED	`'⌂'`
Reserved1	None	`'⌂'`
MsgToken	MQMTOK_NONE	Nulls

Table 156. Initial values of fields in MQGMO for MQGMO (continued)

Field name	Name of constant	Value of constant
ReturnedLength	MQRL_UNDEFINED	-1
Reserved2	None	'⌂'
MsgHandle	MQHM_NONE	0

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQGMO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

   ```
   MQGMO MyGMO = {MQGMO_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQGMO MQGMO;
struct tagMQGMO {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Options;         /* Options that control the action of */
                             MQGET */
  MQLONG    WaitInterval;    /* Wait interval */
  MQLONG    Signal1;         /* Signal */
  MQLONG    Signal2;         /* Signal identifier */
  MQCHAR48  ResolvedQName;   /* Resolved name of destination queue */
  /* Ver:1 */
  MQLONG    MatchOptions;    /* Options controlling selection */
                             criteria used for MQGET */
  MQCHAR    GroupStatus;     /* Flag indicating whether message */
                             retrieved is in a group */
  MQCHAR    SegmentStatus;   /* Flag indicating whether message */
                             retrieved is a segment of a logical */
                             message */
  MQCHAR    Segmentation;    /* Flag indicating whether further */
                             segmentation is allowed for the */
                             message retrieved */
  MQCHAR    Reserved1;       /* Reserved */
  /* Ver:2 */
  MQBYTE16  MsgToken;        /* Message token */
  MQLONG    ReturnedLength;  /* Length of message data returned */
                             (bytes) */
  /* Ver:3 */
  MQLONG    Reserved2;       /* Reserved */
  MQHMSG    MsgHandle;       /* Message handle */
  /* Ver:4 */
};
```

- On z/OS, the *Signal1* field is declared as PMQLONG.

COBOL declaration:

```
**   MQGMO structure
  10 MQGMO.
**    Structure identifier
   15 MQGMO-STRUCID        PIC X(4).
**    Structure version number
   15 MQGMO-VERSION        PIC S9(9) BINARY.
**    Options that control the action of MQGET
   15 MQGMO-OPTIONS        PIC S9(9) BINARY.
**    Wait interval
   15 MQGMO-WAITINTERVAL   PIC S9(9) BINARY.
**    Signal
   15 MQGMO-SIGNAL1        PIC S9(9) BINARY.
**    Signal identifier
   15 MQGMO-SIGNAL2        PIC S9(9) BINARY.
**    Resolved name of destination queue
   15 MQGMO-RESOLVEDQNAME  PIC X(48).
**    Options controlling selection criteria used for MQGET
   15 MQGMO-MATCHOPTIONS   PIC S9(9) BINARY.
**    Flag indicating whether message retrieved is in a group
   15 MQGMO-GROUPSTATUS    PIC X.
**    Flag indicating whether message retrieved is a segment of a
**    logical message
   15 MQGMO-SEGMENTSTATUS  PIC X.
**    Flag indicating whether further segmentation is allowed for the
**    message retrieved
   15 MQGMO-SEGMENTATION   PIC X.
**    Reserved
   15 MQGMO-RESERVED1      PIC X.
**    Message token
   15 MQGMO-MSGTOKEN       PIC X(16).
**    Length of message data returned (bytes)
   15 MQGMO-RETURNEDLENGTH PIC S9(9) BINARY.
**    Reserved
   15 MQGMO-RESERVED2      PIC S9(9) BINARY.
**    Message handle
   15 MQGMO-MSGHANDLE      PIC S9(18) BINARY.
```

- On z/OS, the *Signal1* field is declared as POINTER.

PL/I declaration:

```
dcl
 1 MQGMO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of
                                     MQGET */
  3 WaitInterval   fixed bin(31), /* Wait interval */
  3 Signal1        fixed bin(31), /* Signal */
  3 Signal2        fixed bin(31), /* Signal identifier */
  3 ResolvedQName  char(48),      /* Resolved name of destination
                                     queue */
  3 MatchOptions   fixed bin(31), /* Options controlling selection
                                     criteria used for MQGET */
  3 GroupStatus    char(1),       /* Flag indicating whether message
                                     retrieved is in a group */
  3 SegmentStatus  char(1),       /* Flag indicating whether message
                                     retrieved is a segment of a logical
                                     message */
  3 Segmentation   char(1),       /* Flag indicating whether further
                                     segmentation is allowed for the
                                     message retrieved */
  3 Reserved1      char(1),       /* Reserved */
  3 MsgToken       char(16),      /* Message token */
  3 ReturnedLength fixed bin(31); /* Length of message data returned
```

```
                                   (bytes) */
  3 Reserved2      fixed bin(31); /* Reserved */
  3 MsgHandle      fixed bin(63); /* Message handle */
```

- On z/OS, the *Signal1* field is declared as pointer.

High Level Assembler declaration:

```
MQGMO                 DSECT
MQGMO_STRUCID         DS   CL4   Structure identifier
MQGMO_VERSION         DS   F     Structure version number
MQGMO_OPTIONS         DS   F     Options that control the action of
*                                MQGET
MQGMO_WAITINTERVAL    DS   F     Wait interval
MQGMO_SIGNAL1         DS   F     Signal
MQGMO_SIGNAL2         DS   F     Signal identifier
MQGMO_RESOLVEDQNAME   DS   CL48  Resolved name of destination queue
MQGMO_MATCHOPTIONS    DS   F     Options controlling selection criteria
*                                used for MQGET
MQGMO_GROUPSTATUS     DS   CL1   Flag indicating whether message
*                                retrieved is in a group
MQGMO_SEGMENTSTATUS   DS   CL1   Flag indicating whether message
*                                retrieved is a segment of a logical
*                                message
MQGMO_SEGMENTATION    DS   CL1   Flag indicating whether further
*                                segmentation is allowed for the message
*                                retrieved
MQGMO_RESERVED1       DS   CL1   Reserved
MQGMO_MSGTOKEN        DS   XL16  Message token
MQGMO_RETURNEDLENGTH  DS   F     Length of message data returned (bytes)
MQGMO_RESERVED2       DS   F     Reserved
MQGMO_MSGHANDLE       DS   D     Message handle
MQGMO_LENGTH          EQU  *-MQGMO
                      ORG  MQGMO
MQGMO_AREA            DS   CL(MQGMO_LENGTH)
```

Visual Basic declaration:

```
Type MQGMO
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  Options        As Long      'Options that control the action of MQGET'
  WaitInterval   As Long      'Wait interval'
  Signal1        As Long      'Signal'
  Signal2        As Long      'Signal identifier'
  ResolvedQName  As String*48 'Resolved name of destination queue'
  MatchOptions   As Long      'Options controlling selection criteria'
                              'used for MQGET'
  GroupStatus    As String*1  'Flag indicating whether message'
                              'retrieved is in a group'
  SegmentStatus  As String*1  'Flag indicating whether message'
                              'retrieved is a segment of a logical'
                              'message'
  Segmentation   As String*1  'Flag indicating whether further'
                              'segmentation is allowed for the message'
                              'retrieved'
  Reserved1      As String*1  'Reserved'
  MsgToken       As MQBYTE16  'Message token'
  ReturnedLength As Long      'Length of message data returned (bytes)'
End Type
```

MQIIH - IMS information header:

The following table summarizes the fields in the structure.

Table 157. Fields in MQIIH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQIIH structure	StrucLength
Encoding	Reserved	Encoding
CodedCharSetId	Reserved	CodedCharSetId
Format	MQ format name of data that follows MQIIH	Format
Flags	Flags	Flags
LTermOverride	Logical terminal override	LTermOverride
MFSMapName	Message format services map name	MFSMapName
ReplyToFormat	MQ format name of reply message	ReplyToFormat
Authenticator	RACF™ password or passticket	Authenticator
TranInstanceId	Transaction instance identifier	TranInstanceId
TranState	Transaction state	TranState
CommitMode	Commit mode	CommitMode
SecurityScope	Security scope	SecurityScope
Reserved	Reserved	Reserved

Overview for MQIIH:

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQIIH structure describes the information that must be present at the start of a message sent to the IMS bridge through WebSphere MQ for z/OS.

Format name: MQFMT_IMS.

Character set and encoding: Special conditions apply to the character set and encoding used for the MQIIH structure and application message data:

- Applications that connect to the queue manager that owns the IMS bridge queue must provide an MQIIH structure that is in the character set and encoding of the queue manager. This is because data conversion of the MQIIH structure is not performed in this case.
- Applications that connect to other queue managers can provide an MQIIH structure that is in any of the supported character sets and encodings; the receiving message channel agent connected to the queue manager that owns the IMS bridge queue converts the MQIIH.
- The application message data following the MQIIH structure must be in the same character set and encoding as the MQIIH structure. Do not use the *CodedCharSetId* and *Encoding* fields in the MQIIH structure to specify the character set and encoding of the application message data.

  You must provide a data-conversion exit to convert the application message data if the data is not one of the built-in formats supported by the queue manager.

Fields for MQIIH:

The MQIIH structure contains the following fields; the fields are described in **alphabetical order**:

Authenticator (MQCHAR8):

This is the RACF password or PassTicket. It is optional; if specified, it is used with the user ID in the MQMD security context to build a UTOKEN that is sent to IMS to provide a security context. If it is not specified, the user ID is used without verification. This depends on the setting of the RACF switches, which might require an authenticator to be present.

This is ignored if the first byte is blank or null. The following special value can be used:

MQIAUT_NONE
No authentication.

For the C programming language, the constant MQIAUT_NONE_ARRAY is also defined; this has the same value as MQIAUT_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_AUTHENTICATOR_LENGTH. The initial value of this field is MQIAUT_NONE.

CodedCharSetId (MQLONG):

This is a reserved field; its value is not significant. The initial value of this field is 0.

The Character Set Id for supported structures which follow a MQIIH structure is the same as that of the MQIIH structure itself and taken from any preceding MQ header.

CommitMode (MQCHAR):

This is the IMS commit mode. See the *OTMA Reference* for more information about IMS commit modes. The value must be one of the following:

MQICM_COMMIT_THEN_SEND
Commit then send.

This mode implies double queuing of output, but shorter region occupancy times. Fast-path and conversational transactions cannot run with this mode.

MQICM_SEND_THEN_COMMIT
Send then commit.

Any IMS transaction initiated as a result of a commit mpde of MQICM_SEND_THEN_COMMIT runs in RESPONSE mode regardless of how the transaction is defined in the IMS system definition (MSGTYPE parameter in the TRANSACT macro). This also applies to transactions initiated by means of a transaction switch.

The initial value of this field is MQICM_COMMIT_THEN_SEND.

Encoding (MQLONG):

This is a reserved field; its value is not significant. The initial value of this field is 0.

The Encoding for supported structures which follow a MQIIH structure is the same as that of the MQIIH structure itself and taken from any preceding MQ header.

Flags (MQLONG):

The flags value must be:

MQIIH_NONE
No flags.

MQIIH_PASS_EXPIRATION
The reply message contains:

- The same expiry report options as the request message
- The remaining expiry time from the request message with no adjustment made for the bridge's processing time

If this value is not set, the expiry time is set to *unlimited*.

MQIIH_REPLY_FORMAT_NONE
Sets the MQIIH.Format field of the reply to MQFMT_NONE.

MQIIH_IGNORE_PURG
Sets the TMAMIPRG indicator in the OTMA prefix, which requests that OTMA ignores PURG calls on the TP PCB for CM0 transactions.

MQIIH_CM0_REQUEST_RESPONSE
For Commit Mode 0 (CM0) transactions this flag sets the TMAMHRSP indicator in the OTMA prefix. Setting this indicator requests that OTMA/IMS generate a DFS2082 RESPONSE MODE TRANSACTION TERMINATED WITHOUT REPLY message when the original IMS application program does not reply to the IOPCB nor message switch to another transaction.

The initial value of this field is MQIIH_NONE.

Format (MQCHAR8):

This specifies the MQ format name of the data that follows the MQIIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

LTermOverride (MQCHAR8):

The logical terminal override, placed in the IO PCB field. It is optional; if it is not specified, the TPIPE name is used. It is ignored if the first byte is blank, or null.

The length of this field is given by MQ_LTERM_OVERRIDE_LENGTH. The initial value of this field is 8 blank characters.

MFSMapName (MQCHAR8):

The message format services map name, placed in the IO PCB field. It is optional. On input it represents the MID, on output it represents the MOD. It is ignored if the first byte is blank or null.

The length of this field is given by MQ_MFS_MAP_NAME_LENGTH. The initial value of this field is 8 blank characters.

ReplyToFormat (MQCHAR8):

This is the MQ format name of the reply message that is sent in response to the current message. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

To convert the data in the reply message using MQGMO_CONVERT, specify either MQIIH.replyToFormat=MQFMT_STRING or MQIIH.replyToFormat=MQFMT_IMS_VAR_STRING. For an explanation of the use of these fields, see "Format (MQCHAR8)" on page 1721.

If the default value (MQIIH.replyToFormat=MQFMT_NONE) is used on the request message and the reply message is retrieved using MQGMO_CONVERT then no data conversion is performed.

Reserved (MQCHAR):

This is a reserved field; it must be blank.

SecurityScope (MQCHAR):

This indicates the IMS security processing required. The following values are defined:

MQISS_CHECK
Check security scope: an ACEE is built in the control region, but not in the dependent region.

MQISS_FULL
Full security scope: a cached ACEE is built in the control region and a non-cached ACEE is built in the dependent region. If you use MQISS_FULL, ensure that the user ID for which the ACEE is built has access to the resources used in the dependent region.

If neither MQISS_CHECK nor MQISS_FULL is specified for this field, MQISS_CHECK is assumed.

The initial value of this field is MQISS_CHECK.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQIIH_STRUC_ID
Identifier for the IMS information header structure.

For the C programming language, the constant MQIIH_STRUC_ID_ARRAY is also defined; this has the same value as MQIIH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQIIH_STRUC_ID.

StrucLength (MQLONG):

This is the length of MQIIH structure. The value must be:

MQIIH_LENGTH_1
Length of the IMS information header structure.

The initial value of this field is MQIIH_LENGTH_1.

TranInstanceId (MQBYTE16):

This is the transaction instance identifier. This field is used by output messages from IMS, so is ignored on first input. If you set *TranState* to MQITS_IN_CONVERSATION, this must be provided in the next input, and all subsequent inputs, to enable IMS to correlate the messages to the correct conversation. You can use the following special value:

MQITII_NONE
No transaction instance identifier.

For the C programming language, the constant MQITII_NONE_ARRAY is also defined; this has the same value as MQITII_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_TRAN_INSTANCE_ID_LENGTH. The initial value of this field is MQITII_NONE.

TranState (MQCHAR):

This indicates the IMS conversation state. This is ignored on first input because no conversation exists. On subsequent inputs it indicates whether a conversation is active or not. On output it is set by IMS. The value must be one of the following:

MQITS_IN_CONVERSATION
In conversation.

MQITS_NOT_IN_CONVERSATION
Not in conversation.

MQITS_ARCHITECTED
Return transaction state data in architected form.

This value is used only with the IMS `/DISPLAY TRAN` command. It returns the transaction state data in the IMS architected form instead of character form.

The initial value of this field is MQITS_NOT_IN_CONVERSATION.

Version (MQLONG):

This is the structure version number. The value must be:

MQIIH_VERSION_1
Version number for IMS information header structure.

The following constant specifies the version number of the current version:

MQIIH_CURRENT_VERSION
Current version of IMS information header structure.

The initial value of this field is MQIIH_VERSION_1.

Initial values and language declarations for MQIIH:

Table 158. Initial values of fields in MQIIH for MQIIH

Field name	Name of constant	Value of constant
StrucId	MQIIH_STRUC_ID	'IIH?'
Version	MQIIH_VERSION_1	1
StrucLength	MQIIH_LENGTH_1	84
Encoding	None	0
CodedCharSetId	None	0
Format	MQFMT_NONE	Blanks
Flags	MQIIH_NONE	0
LTermOverride	None	Blanks
MFSMapName	None	Blanks
ReplyToFormat	MQFMT_NONE	Blanks
Authenticator	MQIAUT_NONE	Blanks
TranInstanceId	MQITII_NONE	Nulls
TranState	MQITS_NOT_IN_CONVERSATION	'?'
CommitMode	MQICM_COMMIT_THEN_SEND	'0'
SecurityScope	MQISS_CHECK	'C'
Reserved	None	'?'

Notes:

1. The symbol ? represents a single blank character.
2. In the C programming language, the macro variable MQIIH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

   ```
   MQIIH MyIIH = {MQIIH_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQIIH MQIIH;
struct tagMQIIH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    StrucLength;     /* Length of MQIIH structure */
  MQLONG    Encoding;        /* Reserved */
  MQLONG    CodedCharSetId;  /* Reserved */
  MQCHAR8   Format;          /* MQ format name of data that follows
                                MQIIH */
  MQLONG    Flags;           /* Flags */
  MQCHAR8   LTermOverride;   /* Logical terminal override */
  MQCHAR8   MFSMapName;      /* Message format services map name */
  MQCHAR8   ReplyToFormat;   /* MQ format name of reply message */
  MQCHAR8   Authenticator;   /* RACF password or passticket */
  MQBYTE16  TranInstanceId;  /* Transaction instance identifier */
  MQCHAR    TranState;       /* Transaction state */
  MQCHAR    CommitMode;      /* Commit mode */
  MQCHAR    SecurityScope;   /* Security scope */
  MQCHAR    Reserved;        /* Reserved */
};
```

COBOL declaration:

```
**   MQIIH structure
  10 MQIIH.
**    Structure identifier
   15 MQIIH-STRUCID        PIC X(4).
**    Structure version number
   15 MQIIH-VERSION        PIC S9(9) BINARY.
**    Length of MQIIH structure
   15 MQIIH-STRUCLENGTH    PIC S9(9) BINARY.
**    Reserved
   15 MQIIH-ENCODING       PIC S9(9) BINARY.
**    Reserved
   15 MQIIH-CODEDCHARSETID PIC S9(9) BINARY.
**    MQ format name of data that follows MQIIH
   15 MQIIH-FORMAT         PIC X(8).
**    Flags
   15 MQIIH-FLAGS          PIC S9(9) BINARY.
**    Logical terminal override
   15 MQIIH-LTERMOVERRIDE  PIC X(8).
**    Message format services map name
   15 MQIIH-MFSMAPNAME     PIC X(8).
**    MQ format name of reply message
   15 MQIIH-REPLYTOFORMAT  PIC X(8).
**    RACF password or passticket
   15 MQIIH-AUTHENTICATOR  PIC X(8).
**    Transaction instance identifier
   15 MQIIH-TRANINSTANCEID PIC X(16).
**    Transaction state
   15 MQIIH-TRANSTATE      PIC X.
**    Commit mode
   15 MQIIH-COMMITMODE     PIC X.
**    Security scope
   15 MQIIH-SECURITYSCOPE  PIC X.
**    Reserved
   15 MQIIH-RESERVED       PIC X.
```

PL/I declaration:

```
dcl
 1 MQIIH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Length of MQIIH structure */
  3 Encoding       fixed bin(31), /* Reserved */
  3 CodedCharSetId fixed bin(31), /* Reserved */
  3 Format         char(8),       /* MQ format name of data that follows
                                     MQIIH */
  3 Flags          fixed bin(31), /* Flags */
  3 LTermOverride  char(8),       /* Logical terminal override */
  3 MFSMapName     char(8),       /* Message format services map name */
  3 ReplyToFormat  char(8),       /* MQ format name of reply message */
  3 Authenticator  char(8),       /* RACF password or passticket */
  3 TranInstanceId char(16),      /* Transaction instance identifier */
  3 TranState      char(1),       /* Transaction state */
  3 CommitMode     char(1),       /* Commit mode */
  3 SecurityScope  char(1),       /* Security scope */
  3 Reserved       char(1);       /* Reserved */
```

High Level Assembler declaration:

```
MQIIH                 DSECT
MQIIH_STRUCID         DS   CL4   Structure identifier
MQIIH_VERSION         DS   F     Structure version number
MQIIH_STRUCLENGTH     DS   F     Length of MQIIH structure
MQIIH_ENCODING        DS   F     Reserved
MQIIH_CODEDCHARSETID  DS   F     Reserved
MQIIH_FORMAT          DS   CL8   MQ format name of data that follows
*                                MQIIH
MQIIH_FLAGS           DS   F     Flags
MQIIH_LTERMOVERRIDE   DS   CL8   Logical terminal override
MQIIH_MFSMAPNAME      DS   CL8   Message format services map name
MQIIH_REPLYTOFORMAT   DS   CL8   MQ format name of reply message
MQIIH_AUTHENTICATOR   DS   CL8   RACF password or passticket
MQIIH_TRANINSTANCEID  DS   XL16  Transaction instance identifier
MQIIH_TRANSTATE       DS   CL1   Transaction state
MQIIH_COMMITMODE      DS   CL1   Commit mode
MQIIH_SECURITYSCOPE   DS   CL1   Security scope
MQIIH_RESERVED        DS   CL1   Reserved
*
MQIIH_LENGTH          EQU  *-MQIIH
                      ORG  MQIIH
MQIIH_AREA            DS   CL(MQIIH_LENGTH)
```

Visual Basic declaration:

```
Type MQIIH
  StrucId        As String*4 'Structure identifier'
  Version        As Long     'Structure version number'
  StrucLength    As Long     'Length of MQIIH structure'
  Encoding       As Long     'Reserved'
  CodedCharSetId As Long     'Reserved'
  Format         As String*8 'MQ format name of data that follows MQIIH'
  Flags          As Long     'Flags'
  LTermOverride  As String*8 'Logical terminal override'
  MFSMapName     As String*8 'Message format services map name'
  ReplyToFormat  As String*8 'MQ format name of reply message'
  Authenticator  As String*8 'RACF password or passticket'
  TranInstanceId As MQBYTE16 'Transaction instance identifier'
  TranState      As String*1 'Transaction state'
  CommitMode     As String*1 'Commit mode'
  SecurityScope  As String*1 'Security scope'
  Reserved       As String*1 'Reserved'
End Type
```

MQIMPO - Inquire message property options:

The following table summarizes the fields in the structure. MQIMPO structure - inquire message property options

Table 159. Fields in MQIMPO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options controlling the action of MQINQMP	Options
RequestedEncoding	Encoding into which the inquired property is to be converted	RequestedEncoding
RequestedCCSID	Character set of the inquired property	RequestedCCSID
ReturnedEncoding	Encoding of the returned value	ReturnedEncoding
ReturnedCCSID	Character set of returned value	ReturnedCCSID

Table 159. Fields in MQIMPO (continued)

Field	Description	Topic
Reserved1	Reserved field	ReturnedCCSID
ReturnedName	Name of the inquired property	ReturnedName
TypeString	String representation of the data type of the property	TypeString

Overview for MQIMPO:

The inquire message properties options structure.

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQIMPO structure allows applications to specify options that control how properties of messages are inquired. The structure is an input parameter on the MQINQMP call.

Character set and encoding: Data in MQIMPO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQIMPO:

Inquire message property options structure - fields

The MQIMPO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Inquire message property options structure - Options field

The following options control the action of MQINQMP. You can specify one or more of these options, and if you need more than one, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations of options that are not valid are noted; all other combinations are valid.

Value data options: The following options relate to the processing of the value data when the property is retrieved from the message.

MQIMPO_CONVERT_VALUE
This option requests that the value of the property be converted to conform to the *RequestedCCSID* and *RequestedEncoding* values specified before the MQINQMP call returns the property value in the *Value* area.

- If conversion is successful, the *ReturnedCCSID* and *ReturnedEncoding* fields are set to the same as *RequestedCCSID* and *RequestedEncoding* on return from the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the property value is returned unconverted.

  If the property is a string, the *ReturnedCCSID* and *ReturnedEncoding* fields are set to the character set and encoding of the unconverted string. The completion code is MQCC_WARNING in this case, with reason code MQRC_PROP_VALUE_NOT_CONVERTED. The property cursor is advanced to the returned property.

If the property value expands during conversion, and exceeds the size of the *Value* parameter, the value is returned unconverted, with completion code MQCC_FAILED; the reason code is set to MQRC_PROPERTY_VALUE_TOO_BIG.

The *DataLength* parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

This option also requests that:

- If the property name contains a wildcard, and
- The *ReturnedName* field is initialized with an address or offset for the returned name,

then the returned name is converted to conform to the *RequestedCCSID* and *RequestedEncoding* values.

- If conversion is successful, the *VSCCSID* field of *ReturnedName* and the encoding of the returned name are set to the input value of *RequestedCCSID* and *RequestedEncoding*.
- If conversion fails, but the MQINQMP call otherwise completes without error or warning, the returned name is unconverted. The completion code is MQCC_WARNING in this case, with reason code MQRC_PROP_NAME_NOT_CONVERTED.

  The property cursor is advanced to the returned property.
  MQRC_PROP_VALUE_NOT_CONVERTED is returned if both the value and the name are not converted.

If the returned name expands during conversion, and exceeds the size of the *VSBufsize* field of the *RequestedName*, the returned string is left unconverted, with completion code MQCC_FAILED and the reason code is set to MQRC_PROPERTY_NAME_TOO_BIG.

The *VSLength* field of the MQCHARV structure returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

MQIMPO_CONVERT_TYPE

This option requests that the value of the property be converted from its current data type, into the data type specified on the *Type* parameter of the MQINQMP call.

- If conversion is successful, the *Type* parameter is unchanged on return of the MQINQMP call.
- If conversion fails, but the MQINQMP call otherwise completes without error, the call fails with reason MQRC_PROP_CONV_NOT_SUPPORTED. The property cursor is unchanged.

If the conversion of the data type causes the value to expand during conversion, and the converted value exceeds the size of the *Value* parameter, the value is returned unconverted, with completion code MQCC_FAILED and the reason code is set to MQRC_PROPERTY_VALUE_TOO_BIG.

The *DataLength* parameter of the MQINQMP call returns the length that the property value would have converted to, in order to allow the application to determine the size of the buffer required to accommodate the converted property value. The property cursor is unchanged.

If the value of the *Type* parameter of the MQINQMP call is not valid, the call fails with reason MQRC_PROPERTY_TYPE_ERROR.

If the requested data type conversion is not supported, the call fails with reason MQRC_PROP_CONV_NOT_SUPPORTED. The following data type conversions are supported:

Property data type	Supported target data types
MQTYPE_BOOLEAN	MQTYPE_STRING, MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_BYTE_STRING	MQTYPE_STRING
MQTYPE_INT8	MQTYPE_STRING, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_INT16	MQTYPE_STRING, MQTYPE_INT32, MQTYPE_INT64
MQTYPE_INT32	MQTYPE_STRING, MQTYPE_INT64
MQTYPE_INT64	MQTYPE_STRING
MQTYPE_FLOAT32	MQTYPE_STRING, MQTYPE_FLOAT64
MQTYPE_FLOAT64	MQTYPE_STRING
MQTYPE_STRING	MQTYPE_BOOLEAN, MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32, MQTYPE_INT64, MQTYPE_FLOAT32, MQTYPE_FLOAT64
MQTYPE_NULL	None

The general rules governing the supported conversions are as follows:

- Numeric property values can be converted from one data type to another, provided that no data is lost during the conversion.

  For example, the value of a property with data type MQTYPE_INT32 can be converted into a value with data type MQTYPE_INT64, but cannot be converted into a value with data type MQTYPE_INT16.
- A property value of any data type can be converted into a string.
- A string property value can be converted to any other data type provided the string is formatted correctly for the conversion. If an application attempts to convert a string property value that is not formatted correctly, WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR.
- If an application attempts a conversion that is not supported, WebSphere MQ returns reason code MQRC_PROP_CONV_NOT_SUPPORTED.

The specific rules for converting a property value from one data type to another are as follows:

- When converting an MQTYPE_BOOLEAN property value to a string, the value TRUE is converted to the string "TRUE", and the value false is converted to the string "FALSE".
- When converting an MQTYPE_BOOLEAN property value to a numeric data type, the value TRUE is converted to one, and the value FALSE is converted to zero.
- When converting a string property value to an MQTYPE_BOOLEAN value, the string "TRUE" , or "1" , is converted to TRUE, and the string "FALSE", or "0", is converted to FALSE.

  Note that the terms "TRUE" and "FALSE" are not case sensitive.

  Any other string cannot be converted; WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR.
- When converting a string property value to a value with data type MQTYPE_INT8, MQTYPE_INT16, MQTYPE_INT32 or MQTYPE_INT64, the string must have the following format:

  ```
  [blanks][sign]digits
  ```

  The meanings of the components of the string are as follows:

  blanks Optional leading blank characters

  sign An optional plus sign (+) or minus sign (-) character.

digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

After the sequence of digit characters, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal integer.

WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR if the string is not formatted correctly.

- When converting a string property value to a value with data type MQTYPE_FLOAT32 or MQTYPE_FLOAT64, the string must have the following format:

  ```
  [blanks][sign]digits[.digits][e_char[e_sign]e_digits]
  ```

  The meanings of the components of the string are as follows:

  blanks Optional leading blank characters

  sign An optional plus sign (+) or minus sign (-) character.

  digits A contiguous sequence of digit characters (0-9). At least one digit character must be present.

  e_char An exponent character, which is either "E" or "e".

  e_sign An optional plus sign (+) or minus sign (-) character for the exponent.

  e_digits
  A contiguous sequence of digit characters (0-9) for the exponent. At least one digit character must be present if the string contains an exponent character.

  After the sequence of digit characters, or the optional characters representing an exponent, the string can contain other characters that are not digit characters, but the conversion stops as soon as the first of these characters is reached. The string is assumed to represent a decimal floating point number with an exponent that is a power of 10.

  WebSphere MQ returns reason code MQRC_PROP_NUMBER_FORMAT_ERROR if the string is not formatted correctly.
- When converting a numeric property value to a string, the value is converted to the string representation of the value as a decimal number, not the string containing the ASCII character for that value. For example, the integer 65 is converted to the string "65", not the string "A".
- When converting a byte string property value to a string, each byte is converted to the two hexadecimal characters that represent the byte. For example, the byte array {0xF1, 0x12, 0x00, 0xFF} is converted to the string "F11200FF".

MQIMPO_QUERY_LENGTH
Query the type and length of the property value. The length is returned in the *DataLength* parameter of the MQINQMP call. The property value is not returned.

If a *ReturnedName* buffer is specified, the *VSLength* field of the MQCHARV structure is filled in with the length of the property name. The property name is not returned.

Iteration options: The following options relate to iterating over properties, using a name with a wildcard character

MQIMPO_INQ_FIRST
Inquire on the first property that matches the specified name. After this call, a cursor is established on the property that is returned.

This is the default value.

The MQIMPO_INQ_PROP_UNDER_CURSOR option can subsequently be used with an MQINQMP call, if required, to inquire on the same property again.

Note that there is only one property cursor; therefore, if the property name, specified in the MQINQMP call, changes the cursor is reset.

This option is not valid with either of the following options:

MQIMPO_INQ_NEXT

MQIMPO_INQ_PROP_UNDER_CURSOR

MQIMPO_INQ_NEXT

Inquires on the next property that matches the specified name, continuing the search from the property cursor. The cursor is advanced to the property that is returned.

If this is the first MQINQMP call for the specified name, then the first property that matches the specified name is returned.

The MQIMPO_INQ_PROP_UNDER_CURSOR option can subsequently be used with an MQINQMP call if required, to inquire on the same property again.

If the property under the cursor has been deleted, MQINQMP returns the next matching property following the one that has been deleted.

If a property is added that matches the wildcard, while an iteration is in progress, the property might or might not be returned during the completion of the iteration. The property is returned once the iteration restarts using MQIMPO_INQ_FIRST.

A property matching the wildcard that was deleted, while the iteration was in progress, is not returned subsequent to its deletion.

This option is not valid with either of the following options:

MQIMPO_INQ_FIRST

MQIMPO_INQ_PROP_UNDER_CURSOR

MQIMPO_INQ_PROP_UNDER_CURSOR

Retrieve the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired, using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused, when the message handle is specified in the *MsgHandle* field of the MQGMO on an MQGET call, or when the message handle is specified in *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established, or if the property pointed to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_NOT_AVAILABLE.

This option is not valid with either of the following options:

MQIMPO_INQ_FIRST

MQIMPO_INQ_NEXT

If none of the options previously described is required, the following option can be used:

MQIMPO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values.

MQIMPO_NONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

This is always an input field. The initial value of this field is MQIMPO_INQ_FIRST.

RequestedCCSID (MQLONG):

Inquire message property options structure - RequestedCCSID field

The character set that the inquired property value is to be converted into if the value is a character string. This is also the character set into which the *ReturnedName* is to be converted when MQIMPO_CONVERT_VALUE or MQIMPO_CONVERT_TYPE is specified.

The initial value of this field is MQCCSI_APPL.

RequestedEncoding (MQLONG):

Inquire message property options structure - RequestedEncoding field

This is the encoding into which the inquired property value is to be converted when MQIMPO_CONVERT_VALUE or MQIMPO_CONVERT_TYPE is specified.

The initial value of this field is MQENC_NATIVE.

Reserved1 (MQCHAR):

This is a reserved field. The initial value of this field is a blank character (4 byte field).

ReturnedCCSID (MQLONG):

Inquire message property options structure - ReturnedCCSID field

On output, this is the character set of the value returned if the *Type* parameter of the MQINQMP call is MQTYPE_STRING.

If the MQIMPO_CONVERT_VALUE option is specified and conversion was successful, the *ReturnedCCSID* field, on return, is the same value as the value passed in.

The initial value of this field is zero.

ReturnedEncoding (MQLONG):

Inquire message property options structure - ReturnedEncoding field

On output, this is the encoding of the value returned.

If the MQIMPO_CONVERT_VALUE option is specified and conversion was successful, the *ReturnedEncoding* field, on return, is the same value as the value passed in.

The initial value of this field is MQENC_NATIVE.

ReturnedName (MQCHARV):

Inquire message property options structure - ReturnedName field

The actual name of the inquired property.

On input a string buffer can be passed in using the *VSPtr* or *VSOffset* field of the MQCHARV structure. The length of the string buffer is specified using the *VSBufsize* field of the MQCHARV structure.

On return from the MQINQMP call, the string buffer is completed with the name of the property that was inquired, provided the string buffer was long enough to fully contain the name. The *VSLength* field of the MQCHARV structure is filled in with the length of the property name. The *VSCCSID* field of the MQCHARV structure is filled in to indicate the character set of the returned name, whether or not conversion of the name failed.

This is an input/output field. The initial value of this field is MQCHARV_DEFAULT.

StrucId (MQCHAR4):

Inquire message property options structure - StrucId field

This is the structure identifier. The value must be:

MQIMPO_STRUC_ID
Identifier for inquire message property options structure.

For the C programming language, the constant MQIMPO_STRUC_ID_ARRAY is also defined; this has the same value as MQIMPO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQIMPO_STRUC_ID.

TypeString (MQCHAR8):

Inquire message property options structure - TypeString field

A string representation of the data type of the property.

If the property was specified in an MQRFH2 header and the MQRFH2 `dt` attribute is not recognized, this field can be used to determine the data type of the property. *TypeString* is returned in coded character set 1208 (UTF-8), and is the first eight bytes of the value of the `dt` attribute of the property that failed to be recognized

This is always an output field. The initial value of this field is the null string in the C programming language, and 8 blank characters in other programming languages.

Version (MQLONG):

Inquire message property options structure - Version field

This is the structure version number. The value must be:

MQIMPO_VERSION_1
Version number for inquire message property options structure.

The following constant specifies the version number of the current version:

MQIMPO_CURRENT_VERSION
Current version of inquire message property options structure.

This is always an input field. The initial value of this field is MQIMPO_VERSION_1.

Initial values and language declarations for MQIMPO:

Inquire message property options structure - Initial values

Table 160. Initial values of fields in MQIPMO

Field name	Name of constant	Value of constant
StrucId	MQIMPO_STRUC_ID	'IMPO'
Version	MQIMPO_VERSION_1	1
Options	MQIMPO_INQ_FIRST	
RequestedEncoding	MQENC_NATIVE	
RequestedCCSID	MQCCSI_APPL	
ReturnedEncoding	MQENC_NATIVE	
ReturnedCCSID	0	
Reserved1	0	
ReturnedName	MQCHARV_DEFAULT	
TypeString	Null string or blanks	

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQIMPO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQIMPO MyIMPO = {MQIMPO_DEFAULT};
   ```

C declaration:

Inquire message property options structure - C language declaration

```
typedef struct tagMQIMPO MQIMPO;
struct tagMQIMPO {
  MQCHAR4  StrucId;            /* Structure identifier */
  MQLONG   Version;            /* Structure version number */
  MQLONG   Options;            /* Options that control the action of
                                  MQINQMP */
  MQLONG   RequestedEncoding;  /* Requested encoding of Value */
  MQLONG   RequestedCCSID;     /* Requested character set identifier
                                  of Value */
  MQLONG   ReturnedEncoding;   /* Returned encoding of Value */
  MQLONG   ReturnedCCSID;      /* Returned character set identifier
                                  of Value */
  MQCHAR   Reserved1           /* Reserved field */
  MQCHARV  ReturnedName;       /* Returned property name */
  MQCHAR8  TypeString;         /* Property data type as a string */
};
```

COBOL declaration:

Inquire message property options structure - COBOL language declaration

```
**   MQIMPO structure
  10 MQIMPO.
**    Structure identifier
   15 MQIMPO-STRUCID                PIC X(4).
**    Structure version number
   15 MQIMPO-VERSION                PIC S9(9) BINARY.
**    Options that control the action of MQINQMP
   15 MQIMPO-OPTIONS                PIC S9(9) BINARY.
**    Requested encoding of VALUE
   15 MQIMPO-REQUESTEDENCODING      PIC S9(9) BINARY.
**    Requested character set identifier of VALUE
   15 MQIMPO-REQUESTEDCCSID         PIC S9(9) BINARY.
**    Returned encoding of VALUE
   15 MQIMPO-RETURNEDENCODING       PIC S9(9) BINARY.
**    Returned character set identifier of VALUE
   15 MQIMPO-RETURNEDCCSID          PIC S9(9) BINARY.
**    Reserved field
   15 MQIMPO-RESERVED1
**    Returned property name
   15 MQIMPO-RETURNEDNAME.
**     Address of variable length string
    20 MQIMPO-RETURNEDNAME-VSPTR    POINTER.
**     Offset of variable length string
    20 MQIMPO-RETURNEDNAME-VSOFFSET PIC S9(9) BINARY.
**     CCSID of variable length string
    20 MQIMPO-RETURNEDNAME-VSCCSID  PIC S9(9) BINARY.
**    Property data type as string
   15 MQIMPO-TYPESTRING             PIC S9(9) BINARY.
```

PL/I declaration:

Inquire message property options structure - PL/I language declaration

```
dcl
 1 MQIMPO based,
  3 StrucId            char(4),       /* Structure identifier */
  3 Version            fixed bin(31), /* Structure version number */
  3 Options            fixed bin(31), /* Options that control the
                                         action of MQINQMP */
  3 RequestedEncoding  fixed bin(31), /* Requested encoding of
                                         Value */
  3 RequestedCCSID     fixed bin(31), /* Requested character set
                                         identifier of Value */
  3 ReturnedEncoding   fixed bin(31), /* Returned encoding of
                                         Value */
  3 ReturnedCCSID      fixed bin(31), /* Returned character set
                                         identifier of Value */
  3 Reserved1          fixed bin(31), /* Reserved field */
  3 ReturnedName,                     /* Returned property name */
   5 ReturnedName_VSPtr      pointer,       /* Address of returned
                                               name */
   5 5 ReturnedName_VSOffset fixed bin(31), /* Offset of returned
                                               name */
    5 5 ReturnedName_VSCCSID  fixed bin(31), /* CCSID of returned
                                               name */
  3 TypeString         char(8);       /* Property data type as
                                         string */
```

High Level Assembler declaration:

Inquire message property options structure - Assembler language declaration

```
MQIMPO                       DSECT
MQIMPO_STRUCID               DS   CL4  Structure identifier
MQIMPO_VERSION               DS   F    Structure version number
MQIMPO_OPTIONS               DS   F    Options that control the
*                                      action of MQINQMP
MQIMPO_REQUESTEDENCODING     DS   F    Requested encoding of VALUE
MQIMPO_REQUESTEDCCSID        DS   F    Requested character set
*                                      identifier of VALUE
MQIMPO_RETURNEDENCODING      DS   F    Returned encoding of VALUE
MQIMPO_RETURNEDCCSID         DS   F    Returned character set
*                                      identifier of VALUE
MQIMPO_RESERVED1             DS   F    Reserved field
MQIMPO_RETURNEDNAME          DS   0F   Force fullword alignment
MQIMPO_RETURNEDNAME_VSPTR    DS   F    Address of returned name
MQIMPO_RETURNEDNAME_VSOFFSET DS   F    Offset of returned name
MQIMPO_RETURNEDNAME_VSLENGTH DS   F    Length of returned name
MQIMPO_RETURNEDNAME_VSCCSID  DS   F    CCSID of returned name
MQIMPO_RETURNEDNAME_LENGTH   EQU  *-MQIMPO_RETURNEDNAME
                             ORG  MQIMPO_RETURNEDNAME
MQIMPO_RETURNEDNAME_AREA     DS   CL(MQIMPO_RETURNEDNAME_LENGTH)
*
MQIMPO_TYPESTRING            DS   CL8  Property data type as string
MQIMPO_LENGTH                EQU  *-MQIMPO
MQIMPO_AREA                  DS   CL(MQIMPO_LENGTH)
```

MQMD - Message descriptor:

The following table summarizes the fields in the structure.

Table 161. Fields in MQMD

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Report	Options for report messages	Report
MsgType	Message type	MsgType
Expiry	Message lifetime	MQMD - Expiry field
Feedback	Feedback or reason code	MQMD - Feedback field
Encoding	Numeric encoding of message data	Encoding
CodedCharSetId	Character set identifier of message data	CodedCharSetId
Format	Format name of message data	Format
Priority	Message priority	Priority
Persistence	Message persistence	Persistence
MsgId	Message identifier	MQMD - MsgId field
CorrelId	Correlation identifier	CorrelId
BackoutCount	Backout counter	BackoutCount
ReplyToQ	Name of reply queue	ReplyToQ
ReplyToQMgr	Name of reply queue manager	ReplyToQMgr
UserIdentifier	User identifier	UserIdentifier
AccountingToken	Accounting token	AccountingToken
ApplIdentityData	Application data relating to identity	ApplIdentityData

Table 161. Fields in MQMD (continued)

Field	Description	Topic
PutApplType	Type of application that put the message	PutApplType
PutApplName	Name of application that put the message	PutApplName
PutDate	Date when message was put	PutDate
PutTime	Time when message was put	PutTime
ApplOriginData	Application data relating to origin	ApplOriginData
Note: The remaining fields are ignored if *Version* is less than MQMD_VERSION_2.		
GroupId	Group identifier	GroupId
MsgSeqNumber	Sequence number of logical message within group	MsgSeqNumber
Offset	Offset of data in physical message from start of logical message	Offset
MsgFlags	Message flags	MQMD - MsgFlags field
OriginalLength	Length of original message	OriginalLength

Overview for MQMD:

Availability: All WebSphere MQ systems, plus WebSphere MQ MQI clients connected to these systems.

Purpose: The MQMD structure contains the control information that accompanies the application data when a message travels between the sending and receiving applications. The structure is an input/output parameter on the MQGET, MQPUT, and MQPUT1 calls.

Version: The current version of MQMD is MQMD_VERSION_2. Applications that are intended to be portable between several environments must ensure that the required version of MQMD is supported in all the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQMD that is supported by the environment, but with the initial value of the *Version* field set to MQMD_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

A declaration for the version-1 structure is available with the name MQMD1.

Character set and encoding: Data in MQMD must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE. However, if the application is running as an WebSphere MQ MQI client, the structure must be in the character set and encoding of the client.

If the sending and receiving queue managers use different character sets or encodings, the data in MQMD is converted automatically. It is not necessary for the application to convert the MQMD.

Using different versions of MQMD: A version-2 MQMD is equivalent to using a version-1 MQMD and prefixing the message data with an MQMDE structure. However, if all the fields in the MQMDE structure have their default values, the MQMDE can be omitted. A version-1 MQMD plus MQMDE are used as described:

- On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *Format* field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE.

Note: Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on the MQPUT and MQPUT1 calls. However, the queue manager does *not* return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

- On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a non-default value. The *Format* field in MQMD will have the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

The default values that the queue manager uses for the fields in the MQMDE are the same as the initial values of those fields, shown in Table 166 on page 1763.

When a message is on a transmission queue, some of the fields in MQMD are set to particular values; see "MQXQH - Transmission-queue header" on page 1918 for details.

Message context: Certain fields in MQMD contain the message context. There are two types of message context: *identity context* and *origin context*. Typically:

- Identity context relates to the application that *originally* put the message
- Origin context relates to the application that *most recently* put the message.

These two applications can be the same application, but they can also be different applications (for example, when a message is forwarded from one application to another).

Although identity and origin context typically have the meanings described, the content of both types of context fields in MQMD depends on the MQPMO_*_CONTEXT options that are specified when the message is put. As a result, identity context does not necessarily relate to the application that originally put the message, and origin context does not necessarily relate to the application that most-recently put the message; it depends on the design of the application suite.

The message channel agent (MCA) never alters message context. MCAs that receive messages from remote queue managers use the context option MQPMO_SET_ALL_CONTEXT on the MQPUT or MQPUT1 call. This allows the receiving MCA to preserve exactly the message context that traveled with the message from the sending MCA. However, the result is that the origin context does not relate to either of the MCAs that sent and received the message. The origin context refers to an earlier application that put the message. If all the intermediate applications have passed the message context, the origin context refers to the originating application itself.

In the descriptions, the context fields are described as though they are used as described previously. For more information about message context, see Message context.

Fields for MQMD:

The MQMD structure contains the following fields; the fields are described in **alphabetical order**:

Table 162. Fields in MQMD

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Report	Options for report messages	Report
MsgType	Message type	MsgType
Expiry	Message lifetime	MQMD - Expiry field
Feedback	Feedback or reason code	MQMD - Feedback field
Encoding	Numeric encoding of message data	Encoding

Table 162. Fields in MQMD (continued)

Field	Description	Topic
CodedCharSetId	Character set identifier of message data	CodedCharSetId
Format	Format name of message data	Format
Priority	Message priority	Priority
Persistence	Message persistence	Persistence
MsgId	Message identifier	MQMD - MsgId field
CorrelId	Correlation identifier	CorrelId
BackoutCount	Backout counter	BackoutCount
ReplyToQ	Name of reply queue	ReplyToQ
ReplyToQMgr	Name of reply queue manager	ReplyToQMgr
UserIdentifier	User identifier	UserIdentifier
AccountingToken	Accounting token	AccountingToken
ApplIdentityData	Application data relating to identity	ApplIdentityData
PutApplType	Type of application that put the message	PutApplType
PutApplName	Name of application that put the message	PutApplName
PutDate	Date when message was put	PutDate
PutTime	Time when message was put	PutTime
ApplOriginData	Application data relating to origin	ApplOriginData
Note: The remaining fields are ignored if *Version* is less than MQMD_VERSION_2.		
GroupId	Group identifier	GroupId
MsgSeqNumber	Sequence number of logical message within group	MsgSeqNumber
Offset	Offset of data in physical message from start of logical message	Offset
MsgFlags	Message flags	MQMD - MsgFlags field
OriginalLength	Length of original message	OriginalLength

AccountingToken (MQBYTE32):

This is the accounting token, part of the **identity context** of the message. For more information about message context, see “Overview for MQMD” on page 1706; also see Message context.

AccountingToken allows an application to charge appropriately for work done as a result of the message. The queue manager treats this information as a string of bits and does not check its content.

The queue manager generates this information as follows:

- The first byte of the field is set to the length of the accounting information present in the bytes that follow; this length is in the range zero through 30, and is stored in the first byte as a binary integer.
- The second and subsequent bytes (as specified by the length field) are set to the accounting information appropriate to the environment.
  - On z/OS the accounting information is set to:
    - For z/OS batch, the accounting information from the JES JOB card or from a JES ACCT statement in the EXEC card (comma separators are changed to X'FF'). This information is truncated, if necessary, to 31 bytes.
    - For TSO, the user's account number.
    - For CICS, the LU 6.2 unit of work identifier (UEPUOWDS) (26 bytes).

- For IMS, the 8-character PSB name concatenated with the 16-character IMS recovery token.

– On IBM i, the accounting information is set to the accounting code for the job.
– On UNIX systems, the accounting information is set to the numeric user identifier, in ASCII characters.
– On Windows, the accounting information is set to a Windows security identifier (SID) in a compressed format. The SID uniquely identifies the user identifier stored in the *UserIdentifier* field. When the SID is stored in the *AccountingToken* field, the 6-byte Identifier Authority (located in the third and subsequent bytes of the SID) is omitted. For example, if the Windows SID is 28 bytes long, 22 bytes of SID information are stored in the *AccountingToken* field.

- The last byte (byte 32) of the accounting field is set to the accounting token type (in this case MQACTT_NT_SECURITY_ID, x '0b'):

  MQACTT_CICS_LUOW_ID
  CICS LUOW identifier.

  MQACTT_NT_SECURITY_ID
  Windows security identifier.

  MQACTT_OS400_ACCOUNT_TOKEN
  IBM i accounting token.

  MQACTT_UNIX_NUMERIC_ID
  UNIX systems numeric identifier.

  MQACTT_USER
  User-defined accounting token.

  MQACTT_UNKNOWN
  Unknown accounting-token type.

  The accounting-token type is set to an explicit value only in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems. In other environments, the accounting-token type is set to the value MQACTT_UNKNOWN. In these environments use the *PutApplType* field to deduce the type of accounting token received.
- All other bytes are set to binary zero.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If neither MQPMO_SET_IDENTITY_CONTEXT nor MQPMO_SET_ALL_CONTEXT is specified, this field is ignored on input and is an output-only field. For more information about message context, see Message context.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *AccountingToken* that was transmitted with the message if it was put to a queue. This will be the value of *AccountingToken* that is kept with the message if it is retained (see description of MQPMO_RETAIN in“MQPMO options (MQLONG)” on page 1797 for more details about retained publications) but is not used as the *AccountingToken* when the message is sent as a publication to subscribers since they provide a value to override *AccountingToken* in all publications sent to them. If the message has no context, the field is entirely binary zero.

This is an output field for the MQGET call.

This field is not subject to any translation based on the character set of the queue manager; the field is treated as a string of bits, and not as a string of characters.

The queue manager does nothing with the information in this field. The application must interpret the information if it wants to use the information for accounting purposes.

You can use the following special value for the *AccountingToken* field:

MQACT_NONE
No accounting token is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQACT_NONE_ARRAY is also defined; this has the same value as MQACT_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_ACCOUNTING_TOKEN_LENGTH. The initial value of this field is MQACT_NONE.

ApplIdentityData (MQCHAR32):

This is part of the **identity context** of the message. For more information about message context, see "Overview for MQMD" on page 1706 and Message context.

ApplIdentityData is information that is defined by the application suite, and can be used to provide additional information about the message or its originator. The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If a null character is present, the null and any following characters are converted to blanks by the queue manager. If neither MQPMO_SET_IDENTITY_CONTEXT nor MQPMO_SET_ALL_CONTEXT is specified, this field is ignored on input and is an output-only field. For more information about message context, see Message context.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *ApplIdentityData* that was transmitted with the message if it was put to a queue. This will be the value of *ApplIdentityData* that is kept with the message if it is retained (see description of MQPMO_RETAIN for more details about retained publications) but is not used as the *ApplIdentityData* when the message is sent as a publication to subscribers because they provide a value to override *ApplIdentityData* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by MQ_APPL_IDENTITY_DATA_LENGTH. The initial value of this field is the null string in C, and 32 blank characters in other programming languages.

ApplOriginData (MQCHAR4):

This is part of the **origin context** of the message. For more information about message context, see "Overview for MQMD" on page 1706 and Message context.

ApplOriginData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted.

The queue manager treats this information as character data, but does not define the format of it. When the queue manager generates this information, it is entirely blank.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The length of this field is given by MQ_APPL_ORIGIN_DATA_LENGTH. The initial value of this field is the null string in C, and 4 blank

characters in other programming languages.

BackoutCount (MQLONG):

This is a count of the number of times that the message has been previously returned by the MQGET call as part of a unit of work, and subsequently backed out. It helps the application to detect processing errors that are based on message content. The count excludes MQGET calls that specify any of the MQGMO_BROWSE_* options.

The accuracy of this count is affected by the *HardenGetBackout* queue attribute; see "Attributes for queues" on page 2135.

On z/OS, a value of 255 means that the message has been backed out 255 or more times; the value returned is never greater than 255.

This is an output field for the MQGET call. It is ignored for the MQPUT and MQPUT1 calls. The initial value of this field is `0`.

CodedCharSetId (MQLONG):

This field specifies the character set identifier of character data within the message body.

Note: Character data in MQMD and the other MQ data structures that are parameters on calls must be in the character set of the queue manager. This is defined by the queue manager's *CodedCharSetId* attribute; see "Attributes for the queue manager" on page 2096 for details of this attribute.

If this field is set to MQCCSI_Q_MGR when calling MQGET with MQGMO_CONVERT in the options, the behavior is different between client and server applications. For server applications, the code page used for character conversion is the *CodedCharSetId* of the queue manager; for client applications, the code page used for character conversion is the current locale code page.

For client applications, MQCCSI_Q_MGR is filled in, based on the locale of the client rather than the one on the queue manager. The exception to that rule is when you put a message to an IMS Bridge queue; what is returned, in the *CodedCharSetId* field of MQMD, is the CCSID of the queue manager.

You must not use the following special value:

MQCCSI_APPL
This results in an incorrect value in the *CodedCharSetId* field of the MQMD and causes a return code of MQRC_SOURCE_CCSID_ERROR (or MQRC_FORMAT_ERROR for z/OS) when the message is received using the MQGET call with the MQGMO_CONVERT option.

You can use the following special values:

MQCCSI_Q_MGR
Character data in the message is in the queue manager's character set.

On the MQPUT and MQPUT1 calls, the queue manager changes this value in the MQMD that is sent with the message to the true character set identifier of the queue manager. As a result, the value MQCCSI_Q_MGR is never returned by the MQGET call.

MQCCSI_DEFAULT
The *CodedCharSetId* of the data in the *String* field is defined by the *CodedCharSetId* field in the header structure that precedes the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH is at the start of the message.

MQCCSI_INHERIT
Character data in the message is in the same character set as this structure; this is the queue manager's character set. (For MQMD only, MQCCSI_INHERIT has the same meaning as MQCCSI_Q_MGR).

The queue manager changes this value in the MQMD that is sent with the message to the actual character set identifier of MQMD. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

Do not use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

MQCCSI_EMBEDDED
Character data in the message is in a character set with the identifier that is contained within the message data itself. There can be any number of character set identifiers embedded within the message data, applying to different parts of the data. This value must be used for PCF messages (with a format of MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF) that contain data in a mixture of character sets. Each MQCFST, MQCFSL, and MQCFSF structure contained within the PCF message must have an explicit character set identifier specified and not MQCCSI_DEFAULT.

If a message of format MQFMT_EMBEDDED_PCF is to contain data in a mixture of character sets, do not use MQCCSI_EMBEDDED. Instead set MQEPH_CCSID_EMBEDDED in the Flags field in the MQEPH structure. This is equivalent to setting MQCCSI_EMBEDDED in the preceding structure. Each MQCFST, MQCFSL, and MQCFSF structure contained within the PCF message must then have an explicit character set identifier specified and not MQCCSI_DEFAULT. For more information on the MQEPH structure, see "MQEPH - Embedded PCF header" on page 1650.

Specify this value only on the MQPUT and MQPUT1 calls. If it is specified on the MQGET call, it prevents conversion of the message.

On the MQPUT and MQPUT1 calls, the queue manager changes the values MQCCSI_Q_MGR and MQCCSI_INHERIT in the MQMD that is sent with the message as described above, but does not change the MQMD specified on the MQPUT or MQPUT1 call. No other check is carried out on the value specified.

Applications that retrieve messages must compare this field against the value the application is expecting; if the values differ, the application might need to convert character data in the message.

If you specify the MQGMO_CONVERT option on the MQGET call, this field is an input/output field. The value specified by the application is the coded character set identifier to which to convert the message data if necessary. If conversion is successful or unnecessary, the value is unchanged (except that the value MQCCSI_Q_MGR or MQCCSI_INHERIT is converted to the actual value). If conversion is unsuccessful, the value after the MQGET call represents the coded character set identifier of the unconverted message that is returned to the application.

Otherwise, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQCCSI_Q_MGR.

CorrelId (MQBYTE24):

The CorrelId field is property in the message header that may be used to identify a specific message or group of messages.

This is a byte string that the application can use to relate one message to another, or to relate the message to other work that the application is performing. The correlation identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the correlation identifier is a byte string and not a character string, the correlation identifier is *not* converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, the application can specify any value. The queue manager transmits this value with the message and delivers it to the application that issues the get request for the message.

If the application specifies MQPMO_NEW_CORREL_ID, the queue manager generates a unique correlation identifier which is sent with the message, and also returned to the sending application on output from the MQPUT or MQPUT1 call.

A correlation identifier generated by the queue manager consists of a 3-byte product identifier (AMQ or CSQ in either ASCII or EBCDIC), followed by one reserved byte and a product-specific implementation of a unique string. In WebSphere MQ this product-specific implementation string contains the first 12 characters of the queue-manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters to ensure that message identifiers are unique. The ability to generate a unique string also depends on the system clock not being changed backward. To eliminate the possibility of a message identifier generated by the queue manager duplicating one generated by the application, the application must avoid generating identifiers with initial characters in the range A through I in ASCII or EBCDIC (X'41' through X'49' and X'C1' through X'C9'). However, the application is not prevented from generating identifiers with initial characters in these ranges.

This generated correlation identifier is kept with the message if it is retained, and is used as the correlation identifier when the message is sent as a publication to subscribers who specify MQCI_NONE in the SubCorrelId field in the MQSD passed on the MQSUB call. See MQPMO options for more details about retained publications.

When the queue manager or a message channel agent generates a report message, it sets the *CorrelId* field in the way specified by the *Report* field of the original message, either MQRO_COPY_MSG_ID_TO_CORREL_ID or MQRO_PASS_CORREL_ID. Applications that generate report messages must also do this.

For the MQGET call, *CorrelId* is one of the five fields that can be used to select a particular message to be retrieved from the queue. See the description of the *MsgId* field for details of how to specify values for this field.

Specifying MQCI_NONE as the correlation identifier has the same effect as *not* specifying MQMO_MATCH_CORREL_ID, that is, *any* correlation identifier will match.

If the MQGMO_MSG_UNDER_CURSOR option is specified in the *GetMsgOpts* parameter on the MQGET call, this field is ignored.

On return from an MQGET call, the *CorrelId* field is set to the correlation identifier of the message returned (if any).

The following special values can be used:

MQCI_NONE
No correlation identifier is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQCI_NONE_ARRAY is also defined; this has the same value as MQCI_NONE, but is an array of characters instead of a string.

MQCI_NEW_SESSION
Message is the start of a new session.

This value is recognized by the CICS bridge as indicating the start of a new session, that is, the start of a new sequence of messages.

For the C programming language, the constant MQCI_NEW_SESSION_ARRAY is also defined; this has the same value as MQCI_NEW_SESSION, but is an array of characters instead of a string.

For the MQGET call, this is an input/output field. For the MQPUT and MQPUT1 calls, this is an input field if MQPMO_NEW_CORREL_ID is *not* specified, and an output field if MQPMO_NEW_CORREL_ID *is* specified. The length of this field is given by MQ_CORREL_ID_LENGTH. The initial value of this field is MQCI_NONE.

Note:

You cannot pass the correlation identifier of a publication in a hierarchy. The field is used by the queue manager.

Encoding (MQLONG):

This specifies the numeric encoding of numeric data in the message; it does not apply to numeric data in the MQMD structure itself. The numeric encoding defines the representation used for binary integers, packed-decimal integers, and floating-point numbers.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. The following special value is defined:

MQENC_NATIVE
The encoding is the default for the programming language and machine on which the application is running.

Note: The value of this constant depends on the programming language and environment. For this reason, applications must be compiled using the header, macro, COPY, or INCLUDE files appropriate to the environment in which the application will run.

Applications that put messages usually specify MQENC_NATIVE. Applications that retrieve messages must compare this field against the value MQENC_NATIVE; if the values differ, the application might need to convert numeric data in the message. Use the MQGMO_CONVERT option to request the queue manager to convert the message as part of the processing of the MQGET call. See “Machine encodings” on page 2203 for details of how the *Encoding* field is constructed.

If you specify the MQGMO_CONVERT option on the MQGET call, this field is an input/output field. The value specified by the application is the encoding to which to convert the message data if necessary. If conversion is successful or unnecessary, the value is unchanged. If conversion is unsuccessful, the value after the MQGET call represents the encoding of the unconverted message that is returned to the application.

In other cases, this is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQENC_NATIVE.

Expiry (MQLONG):

This is a period of time expressed in tenths of a second, set by the application that puts the message. The message becomes eligible to be discarded if it has not been removed from the destination queue before this period of time elapses.

The value is decremented to reflect the time that the message spends on the destination queue, and also on any intermediate transmission queues if the put is to a remote queue. It can also be decremented by message channel agents to reflect transmission times, if these are significant. Likewise, an application forwarding this message to another queue might decrement the value if necessary, if it has retained the message for a significant time. However, the expiration time is treated as approximate, and the value need not be decremented to reflect small time intervals.

When the message is retrieved by an application using the MQGET call, the *Expiry* field represents the amount of the original expiry time that still remains.

After a message's expiry time has elapsed, it becomes eligible to be discarded by the queue manager. The message is discarded when a browse or nonbrowse MQGET call occurs that would have returned the message had it not already expired. For example, a nonbrowse MQGET call with the *MatchOptions* field in MQGMO set to MQMO_NONE reading from a FIFO ordered queue discards all the expired messages up to the first unexpired message. With a priority ordered queue, the same call will discard expired messages of higher priority and messages of an equal priority that arrived on the queue before the first unexpired message.

A message that has expired is never returned to an application (either by a browse or a non-browse MQGET call), so the value in the *Expiry* field of the message descriptor after a successful MQGET call is either greater than zero, or the special value MQEI_UNLIMITED.

If a message is put on a remote queue, the message might expire (and be discarded) while it is on an intermediate transmission queue, before the message reaches the destination queue.

A report is generated when an expired message is discarded, if the message specified one of the MQRO_EXPIRATION_* report options. If none of these options is specified, no such report is generated; the message is assumed to be no longer relevant after this time period (perhaps because a later message has superseded it).

For a message put within syncpoint, the expiry interval begins at the time the message is put, not the time the syncpoint is committed. It is possible that the expiry interval can pass before the syncpoint is committed. In this case the message will be discarded at some time after the commit operation and the message will not be returned to an application in response to an MQGET operation.

Any other program that discards messages based on expiry time must also send an appropriate report message if one was requested.

Note:

1. If a message is put with an *Expiry* time of zero or a number greater than 999 999 999, the MQPUT or MQPUT1 call fails with reason code MQRC_EXPIRY_ERROR; no report message is generated in this case.
2. Because a message with an expiry time that has elapsed might not be discarded until later, there might be messages on a queue that have passed their expiry time, and that are not therefore eligible for retrieval. These messages nevertheless count toward the number of messages on the queue for all purposes, including depth triggering.
3. An expiration report is generated, if requested, when the message is discarded, not when it becomes eligible for discarding.

4. Discarding an expired message, and generating an expiration report if requested, are never part of the application's unit of work, even if the message was scheduled for discarding as a result of an MQGET call operating within a unit of work.
5. If a nearly-expired message is retrieved by an MQGET call within a unit of work, and the unit of work is subsequently backed out, the message might become eligible to be discarded before it can be retrieved again.
6. If a nearly-expired message is locked by an MQGET call with MQGMO_LOCK, the message might become eligible to be discarded before it can be retrieved by an MQGET call with MQGMO_MSG_UNDER_CURSOR; reason code MQRC_NO_MSG_UNDER_CURSOR is returned on this subsequent MQGET call if that happens.
7. When a request message with an expiry time greater than zero is retrieved, the application can take one of the following actions when it sends the reply message:
   - Copy the remaining expiry time from the request message to the reply message.
   - Set the expiry time in the reply message to an explicit value greater than zero.
   - Set the expiry time in the reply message to MQEI_UNLIMITED.

   The action to take depends on the design of the application. However, the default action for putting messages to a dead-letter (undelivered-message) queue must be to preserve the remaining expiry time of the message, and to continue to decrement it.
8. Trigger messages are always generated with MQEI_UNLIMITED.
9. A message (normally on a transmission queue) that has a *Format* name of MQFMT_XMIT_Q_HEADER has a second message descriptor within the MQXQH. It therefore has two *Expiry* fields associated with it. The following additional points should be noted in this case:
   - When an application puts a message on a remote queue, the queue manager places the message initially on a local transmission queue, and prefixes the application message data with an MQXQH structure. The queue manager sets the values of the two *Expiry* fields to be the same as that specified by the application.

     If an application puts a message directly on a local transmission queue, the message data must already begin with an MQXQH structure, and the format name must be MQFMT_XMIT_Q_HEADER. In this case, the application need not set the values of these two *Expiry* fields to be the same. (The queue manager checks that the *Expiry* field within the MQXQH contains a valid value, and that the message data is long enough to include it). For an application that can write directly to the transmission queue, the application has to create a transmission queue header with the embedded message descriptor. However, if the expiry value in the message descriptor written to the transmission queue is inconsistent with the value in the embedded message descriptor, an expiry error rejection occurs.
   - When a message with a *Format* name of MQFMT_XMIT_Q_HEADER is retrieved from a queue (whether this is a normal or a transmission queue), the queue manager decrements *both* these *Expiry* fields with the time spent waiting on the queue. No error is raised if the message data is not long enough to include the *Expiry* field in the MQXQH.
   - The queue manager uses the *Expiry* field in the separate message descriptor (that is, not the one in the message descriptor embedded within the MQXQH structure) to test whether the message is eligible for discarding.
   - If the initial values of the two *Expiry* fields are different, the *Expiry* time in the separate message descriptor when the message is retrieved might be greater than zero (so the message is not eligible for discarding), while the time according to the *Expiry* field in the MQXQH has elapsed. In this case the *Expiry* field in the MQXQH is set to zero.
10. The expiry time on a reply message returned from the IMS bridge is unlimited unless MQIIH_PASS_EXPIRATION is set in the Flags field of the MQIIH. See Flags for more information.

The following special value is recognized:

MQEI_UNLIMITED
The message has an unlimited expiration time.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQEI_UNLIMITED.

Feedback (MQLONG):

The Feedback field is used with a message of type MQMT_REPORT to indicate the nature of the report, and is only meaningful with that type of message.

The field can contain one of the MQFB_* values, or one of the MQRC_* values. Feedback codes are grouped as follows:

MQFB_NONE
No feedback provided.

MQFB_SYSTEM_FIRST
Lowest value for system-generated feedback.

MQFB_SYSTEM_LAST
Highest value for system-generated feedback.

The range of system-generated feedback codes MQFB_SYSTEM_FIRST through MQFB_SYSTEM_LAST includes the general feedback codes listed in this topic (MQFB_*), and also the reason codes (MQRC_*) that can occur when the message cannot be put on the destination queue.

MQFB_APPL_FIRST
Lowest value for application-generated feedback.

MQFB_APPL_LAST
Highest value for application-generated feedback.

Applications that generate report messages must not use feedback codes in the system range (other than MQFB_QUIT), unless they want to simulate report messages generated by the queue manager or message channel agent.

On the MQPUT or MQPUT1 calls, the value specified must either be MQFB_NONE, or be within the system range or application range. This is checked whatever the value of *MsgType*.

General feedback codes:

MQFB_COA
Confirmation of arrival on the destination queue (see MQRO_COA).

MQFB_COD
Confirmation of delivery to the receiving application (see MQRO_COD).

MQFB_EXPIRATION
Message was discarded because it had not been removed from the destination queue before its expiry time had elapsed.

MQFB_PAN
Positive action notification (see MQRO_PAN).

MQFB_NAN
Negative action notification (see MQRO_NAN).

MQFB_QUIT
End application.

This can be used by a workload scheduling program to control the number of instances of an application program that are running. Sending an MQMT_REPORT message with this feedback code to an instance of the application program indicates to that instance that it should stop processing. However, adherence to this convention is a matter for the application; it is not enforced by the queue manager.

Channel feedback codes:

MQFB_CHANNEL_COMPLETED
A channel ended normally.

MQFB_CHANNEL_FAIL
A channel ended abnormally and goes into STOPPED state.

MQFB_CHANNEL_FAIL_RETRY
A channel ended abnormally and goes into RETRY state.

IMS-bridge feedback codes

These codes are used when an unexpected IMS-OTMA sense code is received. The sense code or, when the sense code is 0x1A the reason code associated with that sense code, is indicated in the *Feedback*.

1. For *Feedback* codes in range MQFB_IMS_FIRST (300) through MQFB_IMS_LAST (399), a sense code other than 0x1A was received. The *sense code* is given by the expression (*Feedback* - MQFB_IMS_FIRST+1)
2. For *Feedback* codes in range MQFB_IMS_NACK_1A_REASON_FIRST (600) through MQFB_IMS_NACK_1A_REASON_LAST (855), a sense code of 0x1A was received. The *reason code* associated with the sense code is given by the expression (*Feedback* - MQFB_IMS_NACK_1A_REASON_FIRST)

The meaning of the IMS-OTMA sense codes and corresponding reason codes are described in *Open Transaction Manager Access Guide and Reference*.

The following feedback codes can be generated by the IMS bridge:

MQFB_DATA_LENGTH_ZERO
A segment length was zero in the application data of the message.

MQFB_DATA_LENGTH_NEGATIVE
A segment length was negative in the application data of the message.

MQFB_DATA_LENGTH_TOO_BIG
A segment length was too large in the application data of the message.

MQFB_BUFFER_OVERFLOW
The value of one of the length fields would cause the data to overflow the message buffer.

MQFB_LENGTH_OFF_BY_ONE
The value of one of the length fields was 1 byte too short.

MQFB_IIH_ERROR
The *Format* field in MQMD specifies MQFMT_IMS, but the message does not begin with a valid MQIIH structure.

MQFB_NOT_AUTHORIZED_FOR_IMS
The user ID contained in the message descriptor MQMD, or the password contained in the *Authenticator* field in the MQIIH structure, failed the validation performed by the IMS bridge. As a result the message was not passed to IMS.

MQFB_IMS_ERROR
An unexpected error was returned by IMS. Consult the WebSphere MQ error log on the system on which the IMS bridge resides for more information about the error.

MQFB_IMS_FIRST
When the IMS-OTMA sense code is not 0x1A, IMS-generated feedback codes are in the range MQFB_IMS_FIRST (300) through MQFB_IMS_LAST (399). The IMS-OTMA sense code itself is *Feedback* minus MQFB_IMS_ERROR.

MQFB_IMS_LAST
Highest value for IMS-generated feedback when the sense code is not 0x1A.

MQFB_IMS_NACK_1A_REASON_FIRST
When the sense code is 0x1A, IMS-generated feedback codes are in the range MQFB_IMS_NACK_1A_REASON_FIRST (600) through MQFB_IMS_NACK_1A_REASON_LAST (855).

MQFB_IMS_NACK_1A_REASON_LAST
Highest value for IMS-generated feedback when the sense code is 0x1A

CICS-bridge feedback codes: The following feedback codes can be generated by the CICS bridge:

MQFB_CICS_APPL_ABENDED
The application program specified in the message abnormally ended. This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_APPL_NOT_STARTED
The EXEC CICS LINK for the application program specified in the message failed. This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_BRIDGE_FAILURE
CICS bridge terminated abnormally without completing normal error processing.

MQFB_CICS_CCSID_ERROR
Character set identifier not valid.

MQFB_CICS_CIH_ERROR
CICS information header structure missing or not valid.

MQFB_CICS_COMMAREA_ERROR
Length of CICS COMMAREA not valid.

MQFB_CICS_CORREL_ID_ERROR
Correlation identifier not valid.

MQFB_CICS_DLQ_ERROR
The CICS bridge task was unable to copy a reply to this request to the dead-letter queue. The request was backed out.

MQFB_CICS_ENCODING_ERROR
Encoding not valid.

MQFB_CICS_INTERNAL_ERROR
CICS bridge encountered an unexpected error.

This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_NOT_AUTHORIZED
User identifier not authorized or password not valid.

This feedback code occurs only in the *Reason* field of the MQDLH structure.

MQFB_CICS_UOW_BACKED_OUT
The unit of work was backed out, for one of the following reasons:

- A failure was detected while processing another request within the same unit of work.
- A CICS abend occurred while the unit of work was in progress.

MQFB_CICS_UOW_ERROR
Unit-of-work control field *UOWControl* not valid.

Trace-route message feedback codes:

MQFB_ACTIVITY
Used with the MQFMT_EMBEDDED_PCF format to allow the option of user data following activity reports.

MQFB_MAX_ACTIVITIES
Returned when the trace-route message is discarded because the number of activities the message has been involved in exceeds the maximum activities limit.

MQFB_NOT_FORWARDED
Returned when the trace-route message is discarded because it is about to be sent to a remote queue manager that does not support trace-route messages.

MQFB_NOT_DELIVERED
Returned when the trace-route message is discarded because it is about to be put on a local queue.

MQFB_UNSUPPORTED_FORWARDING
Returned when the trace-route message is discarded because a value in the forwarding parameter is unrecognized, and is in the rejected bit mask.

MQFB_UNSUPPORTED_DELIVERY
Returned when the trace-route message is discarded because a value in the delivery parameter is unrecognized, and is in the rejected bit mask.

WebSphere MQ reason codes: For exception report messages, *Feedback* contains an WebSphere MQ reason code. Among possible reason codes are:

MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue.

MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.

MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Queue does not support persistent messages.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
(2031, X'7EF') Message length greater than maximum for queue manager.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

For a full list of reason codes, see:
- For WebSphere MQ for z/OS, see API reason codes.
- For all other platforms, see API completion and reason codes.

.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQFB_NONE.

Format (MQCHAR8):

This is a name that the sender of the message uses to indicate to the receiver the nature of the data in the message. Any characters that are in the character set of the queue manager can be specified for the name, but you must restrict the name to the following:

- Uppercase A through Z
- Numeric digits 0 through 9

If other characters are used, it might not be possible to translate the name between the character sets of the sending and receiving queue managers.

Pad the name with blanks to the length of the field, or use a null character to terminate the name before the end of the field; the null and any subsequent characters are treated as blanks. Do not specify a name with leading or embedded blanks. For the MQGET call, the queue manager returns the name padded with blanks to the length of the field.

The queue manager does not check that the name complies with the recommendations described above.

Names beginning MQ in upper, lower, and mixed case have meanings that are defined by the queue manager; do not use names beginning with these letters for your own formats. The queue manager built-in formats are:

MQFMT_NONE
The nature of the data is undefined: the data cannot be converted when the message is retrieved from a queue using the MQGMO_CONVERT option.

If you specify MQGMO_CONVERT on the MQGET call, and the character set or encoding of data in the message differs from that specified in the *MsgDesc* parameter, the message is returned with the following completion and reason codes (assuming no other errors):

- Completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR if the MQFMT_NONE data is at the beginning of the message.
- Completion code MQCC_OK and reason code MQRC_NONE if the MQFMT_NONE data is at the end of the message (that is, preceded by one or more MQ header structures). The MQ header structures are converted to the requested character set and encoding in this case.

For the C programming language, the constant MQFMT_NONE_ARRAY is also defined; this has the same value as MQFMT_NONE, but is an array of characters instead of a string.

MQFMT_ADMIN
The message is a command-server request or reply message in programmable command format (PCF). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call. See Using Programmable Command Formats for more information about using programmable command format messages.

For the C programming language, the constant MQFMT_ADMIN_ARRAY is also defined; this has the same value as MQFMT_ADMIN, but is an array of characters instead of a string.

MQFMT_CICS
The message data begins with the CICS information header MQCIH, followed by the application data. The format name of the application data is given by the *Format* field in the MQCIH structure.

On z/OS, specify the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_CICS.

For the C programming language, the constant MQFMT_CICS_ARRAY is also defined; this has the same value as MQFMT_CICS, but is an array of characters instead of a string.

MQFMT_COMMAND_1
The message is an MQSC command-server reply message containing the object count, completion code, and reason code. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_COMMAND_1_ARRAY is also defined; this has the same value as MQFMT_COMMAND_1, but is an array of characters instead of a string.

MQFMT_COMMAND_2
The message is an MQSC command-server reply message containing information about the objects requested. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_COMMAND_2_ARRAY is also defined; this has the same value as MQFMT_COMMAND_2, but is an array of characters instead of a string.

MQFMT_DEAD_LETTER_HEADER
The message data begins with the dead-letter header MQDLH. The data from the original message immediately follows the MQDLH structure. The format name of the original message data is given by the *Format* field in the MQDLH structure; see "MQDLH - Dead-letter header" on page 1636 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

COA and COD reports are not generated for messages that have a *Format* of MQFMT_DEAD_LETTER_HEADER.

For the C programming language, the constant MQFMT_DEAD_LETTER_HEADER_ARRAY is also defined; this has the same value as MQFMT_DEAD_LETTER_HEADER, but is an array of characters instead of a string.

MQFMT_DIST_HEADER
The message data begins with the distribution-list header MQDH; this includes the arrays of MQOR and MQPMR records. The distribution-list header can be followed by additional data. The format of the additional data (if any) is given by the *Format* field in the MQDH structure; see "MQDH - Distribution header" on page 1629 for details of this structure. Messages with format MQFMT_DIST_HEADER can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

This format is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

For the C programming language, the constant MQFMT_DIST_HEADER_ARRAY is also defined; this has the same value as MQFMT_DIST_HEADER, but is an array of characters instead of a string.

MQFMT_EMBEDDED_PCF
Format for a trace-route message, provided that the PCF command value is set to MQCMD_TRACE_ROUTE. Using this format allows user data to be sent along with the trace-route message, provided that their applications can cope with preceding PCF parameters.

The PCF header **must** be the first header, or the message will not be treated as a trace-route message. This means that the message cannot be in a group, and that trace-route messages cannot be segmented. If a trace-route message is sent in a group the message is rejected with reason code MQRC_MSG_NOT_ALLOWED_IN_GROUP.

Note that MQFMT_ADMIN can also be used for the format of a trace-route message, but in this case no user data can be sent along with the trace-route message.

MQFMT_EVENT
The message is an MQ event message that reports an event that occurred. Event messages have

the same structure as programmable commands; see PCF command messages for more information about this structure, and Event monitoring for information about events.

Version-1 event messages can be converted in all environments if the MQGMO_CONVERT option is specified on the MQGET call. Version-2 event messages can be converted only on z/OS.

For the C programming language, the constant MQFMT_EVENT_ARRAY is also defined; this has the same value as MQFMT_EVENT, but is an array of characters instead of a string.

MQFMT_IMS
The message data begins with the IMS information header MQIIH, which is followed by the application data. The format name of the application data is given by the *Format* field in the MQIIH structure.

For details of how MQIIH structure is handled when using MQGET with MQGMO_CONVERT, see "Format (MQCHAR8)" on page 1690 and "ReplyToFormat (MQCHAR8)" on page 1691.

For the C programming language, the constant MQFMT_IMS_ARRAY is also defined; this has the same value as MQFMT_IMS, but is an array of characters instead of a string.

MQFMT_IMS_VAR_STRING
The message is an IMS variable string, which is a string of the form `llzzccc`, where:

ll is a 2-byte length field specifying the total length of the IMS variable string item. This length is equal to the length of `ll` (2 bytes), plus the length of `zz` (2 bytes), plus the length of the character string itself. `ll` is a 2-byte binary integer in the encoding specified by the *Encoding* field.

zz is a 2-byte field containing flags that are significant to IMS. `zz` is a byte string consisting of two MQBYTE fields, and is transmitted without change from sender to receiver (that is, `zz` is not subject to any conversion).

ccc is a variable-length character string containing `ll-4` characters. `ccc` is in the character set specified by the *CodedCharSetId* field.

On z/OS, the message data can consist of a sequence of IMS variable strings butted together, with each string being of the form `llzzccc`. There must be no bytes skipped between successive IMS variable strings. This means that if the first string has an odd length, the second string will be misaligned, that is, it will not begin on a boundary that is a multiple of two. Take care when constructing such strings on machines that require alignment of elementary data types.

Use the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_IMS_VAR_STRING.

For the C programming language, the constant MQFMT_IMS_VAR_STRING_ARRAY is also defined; this has the same value as MQFMT_IMS_VAR_STRING, but is an array of characters instead of a string.

MQFMT_MD_EXTENSION
The message data begins with the message-descriptor extension MQMDE, and is optionally followed by other data (usually the application message data). The format name, character set, and encoding of the data that follow the MQMDE are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMDE. See "MQMDE - Message descriptor extension" on page 1758 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_MD_EXTENSION_ARRAY is also defined; this has the same value as MQFMT_MD_EXTENSION, but is an array of characters instead of a string.

MQFMT_PCF
The message is a user-defined message that conforms to the structure of a programmable command format (PCF) message. Messages of this format can be converted if the

MQGMO_CONVERT option is specified on the MQGET call. See Using Programmable Command Formats for more information about using programmable command format messages.

For the C programming language, the constant MQFMT_PCF_ARRAY is also defined; this has the same value as MQFMT_PCF, but is an array of characters instead of a string.

MQFMT_REF_MSG_HEADER
The message data begins with the reference message header MQRMH, and is optionally followed by other data. The format name, character set, and encoding of the data is given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRMH. See "MQRMH - Reference message header" on page 1841 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

This format is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

For the C programming language, the constant MQFMT_REF_MSG_HEADER_ARRAY is also defined; this has the same value as MQFMT_REF_MSG_HEADER, but is an array of characters instead of a string.

MQFMT_RF_HEADER
The message data begins with the rules and formatting header MQRFH, and is optionally followed by other data. The format name, character set, and encoding of the data (if any) are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRFH. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_RF_HEADER_ARRAY is also defined; this has the same value as MQFMT_RF_HEADER, but is an array of characters instead of a string.

MQFMT_RF_HEADER_2
The message data begins with the version-2 rules and formatting header MQRFH2, and is optionally followed by other data. The format name, character set, and encoding of the optional data (if any) are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQRFH2. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_RF_HEADER_2_ARRAY is also defined; this has the same value as MQFMT_RF_HEADER_2, but is an array of characters instead of a string.

MQFMT_STRING
The application message data can be either an SBCS string (single-byte character set), or a DBCS string (double-byte character set). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_STRING_ARRAY is also defined; this has the same value as MQFMT_STRING, but is an array of characters instead of a string.

MQFMT_TRIGGER
The message is a trigger message, described by the MQTM structure; see "MQTM - Trigger message" on page 1897 for details of this structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

For the C programming language, the constant MQFMT_TRIGGER_ARRAY is also defined; this has the same value as MQFMT_TRIGGER, but is an array of characters instead of a string.

MQFMT_WORK_INFO_HEADER
The message data begins with the work information header MQWIH, which is followed by the application data. The format name of the application data is given by the *Format* field in the MQWIH structure.

On z/OS, specify the MQGMO_CONVERT option on the MQGET call to convert the *user data* in messages that have format MQFMT_WORK_INFO_HEADER. However, the MQWIH structure itself is always returned in the queue-manager's character set and encoding (that is, the MQWIH structure is converted whether or not the MQGMO_CONVERT option is specified).

For the C programming language, the constant MQFMT_WORK_INFO_HEADER_ARRAY is also defined; this has the same value as MQFMT_WORK_INFO_HEADER, but is an array of characters instead of a string.

MQFMT_XMIT_Q_HEADER
The message data begins with the transmission queue header MQXQH. The data from the original message immediately follows the MQXQH structure. The format name of the original message data is given by the *Format* field in the MQMD structure, which is part of the transmission queue header MQXQH. See "MQXQH - Transmission-queue header" on page 1918 for details of this structure.

COA and COD reports are not generated for messages that have a *Format* of MQFMT_XMIT_Q_HEADER.

For the C programming language, the constant MQFMT_XMIT_Q_HEADER_ARRAY is also defined; this has the same value as MQFMT_XMIT_Q_HEADER, but is an array of characters instead of a string.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24):

This is a byte string that is used to identify the particular message group or logical message to which the physical message belongs. *GroupId* is also used if segmentation is allowed for the message. In all these cases, *GroupId* has a non-null value, and one or more of the following flags is set in the *MsgFlags* field:

- MQMF_MSG_IN_GROUP
- MQMF_LAST_MSG_IN_GROUP
- MQMF_SEGMENT
- MQMF_LAST_SEGMENT
- MQMF_SEGMENTATION_ALLOWED

If none of these flags is set, *GroupId* has the special null value MQGI_NONE.

The application does not need to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_GROUP_ID is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *GroupId* is set to an appropriate value.

Message groups and segments can be processed correctly only if the group identifier is unique. For this reason, *applications must not generate their own group identifiers*; instead, applications must do one of the following:

- If MQPMO_LOGICAL_ORDER is specified, the queue manager automatically generates a unique group identifier for the first message in the group or segment of the logical message, and uses that group identifier for the remaining messages in the group or segments of the logical message, so the application does not need to take any special action. This is the recommended procedure.
- If MQPMO_LOGICAL_ORDER is *not* specified, the application must request the queue manager to generate the group identifier, by setting *GroupId* to MQGI_NONE on the first MQPUT or MQPUT1 call for a message in the group or segment of the logical message. The group identifier returned by the

queue manager on output from that call must then be used for the remaining messages in the group or segments of the logical message. If a message group contains segmented messages, the same group identifier must be used for all segments and messages in the group.

When MQPMO_LOGICAL_ORDER is not specified, messages in groups and segments of logical messages can be put in any order (for example, in reverse order), but the group identifier must be allocated by the *first* MQPUT or MQPUT1 call that is issued for any of those messages.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Physical order on a queue. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message if the object opened is a single queue and not a distribution list, but leaves it unchanged if the object opened is a distribution list. In the latter case, if the application needs to know the group identifiers generated, the application must provide MQPMR records containing the *GroupId* field.

On input to the MQGET call, the queue manager uses the value described in Table 154 on page 1675. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The following special value is defined:

MQGI_NONE
No group identifier specified.

The value is binary zero for the length of the field. This is the value that is used for messages that are not in groups, not segments of logical messages, and for which segmentation is not allowed.

For the C programming language, the constant MQGI_NONE_ARRAY is also defined; this has the same value as MQGI_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_GROUP_ID_LENGTH. The initial value of this field is MQGI_NONE. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgFlags (MQLONG):

MsgFlags are flags that specify attributes of the message, or control its processing.

MsgFlags are divided into the following categories:

- Segmentation flags
- Status flags

Segmentation flags: When a message is too big for a queue, an attempt to put the message on the queue typically fails. Segmentation is a technique whereby the queue manager or application splits the message into smaller pieces called segments, and places each segment on the queue as a separate physical message. The application that retrieves the message can either retrieve the segments one by one, or request the queue manager to reassemble the segments into a single message that is returned by the MQGET call. The latter is achieved by specifying the MQGMO_COMPLETE_MSG option on the MQGET call, and supplying a buffer that is big enough to accommodate the complete message. (See "MQGMO - Get-message options" on page 1655 for details of the MQGMO_COMPLETE_MSG option.) A message can be segmented at the sending queue manager, at an intermediate queue manager, or at the destination queue manager.

You can specify one of the following to control the segmentation of a message:

MQMF_SEGMENTATION_INHIBITED
This option prevents the message being broken into segments by the queue manager. If specified for a message that is already a segment, this option prevents the segment being broken into smaller segments.

The value of this flag is binary zero. This is the default.

MQMF_SEGMENTATION_ALLOWED
This option allows the message to be broken into segments by the queue manager. If specified for a message that is already a segment, this option allows the segment to be broken into smaller segments. MQMF_SEGMENTATION_ALLOWED can be set without either MQMF_SEGMENT or MQMF_LAST_SEGMENT being set.

- On z/OS, the queue manager does not support the segmentation of messages. If a message is too big for the queue, the MQPUT or MQPUT1 call fails with reason code MQRC_MSG_TOO_BIG_FOR_Q. However, the MQMF_SEGMENTATION_ALLOWED option can still be specified, and allows the message to be segmented at a remote queue manager.

When the queue manager segments a message, the queue manager turns on the MQMF_SEGMENT flag in the copy of the MQMD that is sent with each segment, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call. For the last segment in the logical message, the queue manager also turns on the MQMF_LAST_SEGMENT flag in the MQMD that is sent with the segment.

Note: Take care when putting messages with MQMF_SEGMENTATION_ALLOWED but without MQPMO_LOGICAL_ORDER. If the message is:

- Not a segment, and
- Not in a group, and
- Not being forwarded,

the application must reset the *GroupId* field to MQGI_NONE before *each* MQPUT or MQPUT1 call, so that the queue manager can generate a unique group identifier for each message. If this is not done, unrelated messages can have the same group identifier, which might lead to incorrect processing subsequently. See the descriptions of the *GroupId* field and the MQPMO_LOGICAL_ORDER option for more information about when to reset the *GroupId* field.

The queue manager splits messages into segments as necessary so that the segments (plus any required header data) fit on the queue. However, there is a lower limit for the size of a segment generated by the queue manager, and only the last segment created from a message can be smaller than this limit (the lower limit for the size of an application-generated segment is one byte). Segments generated by the queue manager might be of unequal length. The queue-manager processes the message as follows:

- User-defined formats are split on boundaries that are multiples of 16 bytes; the queue manager does not generate segments that are smaller than 16 bytes (other than the last segment).
- Built-in formats other than MQFMT_STRING are split at points appropriate to the nature of the data present. However, the queue manager never splits a message in the middle of an WebSphere MQ header structure. This means that a segment containing a single MQ header structure cannot be split further by the queue manager, and as a result the minimum possible segment size for that message is greater than 16 bytes.

  The second or later segment generated by the queue manager begins with one of the following:
  - An MQ header structure
  - The start of the application message data
  - Part of the way through the application message data
- MQFMT_STRING is split without regard for the nature of the data present (SBCS, DBCS, or mixed SBCS/DBCS). When the string is DBCS or mixed SBCS/DBCS, this might result in segments that cannot be converted from one character set to another. The queue manager never splits MQFMT_STRING messages into segments that are smaller than 16 bytes (other than the last segment).
- The queue manager sets the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMD of each segment to describe correctly the data present at the *start* of the segment; the format name is either the name of a built-in format, or the name of a user-defined format.

- The *Report* field in the MQMD of segments with *Offset* greater than zero is modified. For each report type, if the report option is MQRO_*_WITH_DATA, but the segment cannot contain any of the first 100 bytes of user data (that is, the data following any WebSphere MQ header structures that may be present), the report option is changed to MQRO_*.

The queue manager follows the above rules, but otherwise splits messages unpredictably; do not make assumptions about where a message is split.

For *persistent* messages, the queue manager can perform segmentation only within a unit of work:

- If the MQPUT or MQPUT1 call is operating within a user-defined unit of work, that unit of work is used. If the call fails during the segmentation process, the queue manager removes any segments that were placed on the queue as a result of the failing call. However, the failure does not prevent the unit of work being committed successfully.
- If the call is operating outside a user-defined unit of work, and there is no user-defined unit of work in existence, the queue manager creates a unit of work just for the duration of the call. If the call is successful, the queue manager commits the unit of work automatically. If the call fails, the queue manager backs out the unit of work.
- If the call is operating outside a user-defined unit of work, but a user-defined unit of work exists, the queue manager cannot perform segmentation. If the message does not require segmentation, the call can still succeed. But if the message requires segmentation, the call fails with reason code MQRC_UOW_NOT_AVAILABLE.

For *nonpersistent* messages, the queue manager does not require a unit of work to be available in order to perform segmentation.

Take special care when converting data in messages that might be segmented:

- If the receiving application converts data on the MQGET call, and specifies the MQGMO_COMPLETE_MSG option, the data-conversion exit is passed the complete message for the exit to convert, and the fact that the message was segmented is apparent to the exit.
- If the receiving application retrieves one segment at a time, the data-conversion exit is invoked to convert one segment at a time. The exit must therefore convert the data in a segment independently of the data in any of the other segments.

  If the nature of the data in the message is such that arbitrary segmentation of the data on 16-byte boundaries might result in segments that cannot be converted by the exit, or the format is MQFMT_STRING and the character set is DBCS or mixed SBCS/DBCS, the sending application must create and put the segments, specifying MQMF_SEGMENTATION_INHIBITED to suppress further segmentation. In this way, the sending application can ensure that each segment contains sufficient information to allow the data-conversion exit to convert the segment successfully.
- If sender conversion is specified for a sending message channel agent (MCA), the MCA converts only messages that are not segments of logical messages; the MCA never attempts to convert messages that are segments.

This flag is an input flag on the MQPUT and MQPUT1 calls, and an output flag on the MQGET call. On the latter call, the queue manager also echoes the value of the flag to the *Segmentation* field in MQGMO.

The initial value of this flag is MQMF_SEGMENTATION_INHIBITED.

Status flags: These are flags that indicate whether the physical message belongs to a message group, is a segment of a logical message, both, or neither. One or more of the following can be specified on the MQPUT or MQPUT1 call, or returned by the MQGET call:

MQMF_MSG_IN_GROUP
Message is a member of a group.

MQMF_LAST_MSG_IN_GROUP
Message is the last logical message in a group.

If this flag is set, the queue manager turns on MQMF_MSG_IN_GROUP in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

It is valid for a group to consist of only one logical message. If this is the case, MQMF_LAST_MSG_IN_GROUP is set, but the *MsgSeqNumber* field has the value one.

MQMF_SEGMENT
Message is a segment of a logical message.

When MQMF_SEGMENT is specified without MQMF_LAST_SEGMENT, the length of the application message data in the segment (*excluding* the lengths of any WebSphere MQ header structures that might be present) must be at least one. If the length is zero, the MQPUT or MQPUT1 call fails with reason code MQRC_SEGMENT_LENGTH_ZERO.

On z/OS, this option is not supported if the message is being put on a queue that has an index type of MQIT_GROUP_ID.

MQMF_LAST_SEGMENT
Message is the last segment of a logical message.

If this flag is set, the queue manager turns on MQMF_SEGMENT in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

A logical message can consist of only one segment. If so, MQMF_LAST_SEGMENT is set, but the *Offset* field has the value zero.

When MQMF_LAST_SEGMENT is specified, the length of the application message data in the segment (*excluding* the lengths of any header structures that might be present) can be zero.

On z/OS, this option is not supported if the message is being put on a queue that has an index type of MQIT_GROUP_ID.

The application must ensure that these flags are set correctly when putting messages. If MQPMO_LOGICAL_ORDER is specified, or was specified on the preceding MQPUT call for the queue handle, the settings of the flags must be consistent with the group and segment information retained by the queue manager for the queue handle. The following conditions apply to *successive* MQPUT calls for the queue handle when MQPMO_LOGICAL_ORDER is specified:

- If there is no current group or logical message, all these flags (and combinations of them) are valid.
- Once MQMF_MSG_IN_GROUP has been specified, it must remain on until MQMF_LAST_MSG_IN_GROUP is specified. The call fails with reason code MQRC_INCOMPLETE_GROUP if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified, it must remain on until MQMF_LAST_SEGMENT is specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.
- Once MQMF_SEGMENT has been specified without MQMF_MSG_IN_GROUP, MQMF_MSG_IN_GROUP must remain *off* until after MQMF_LAST_SEGMENT has been specified. The call fails with reason code MQRC_INCOMPLETE_MSG if this condition is not satisfied.

Physical order on a queue shows the valid combinations of the flags, and the values used for various fields.

These flags are input flags on the MQPUT and MQPUT1 calls, and output flags on the MQGET call. On the latter call, the queue manager also echoes the values of the flags to the *GroupStatus* and *SegmentStatus* fields in MQGMO.

You cannot use grouped or segmented messages with Publish/Subscribe.

Default flags: The following can be specified to indicate that the message has default attributes:

MQMF_NONE
No message flags (default message attributes).

This inhibits segmentation, and indicates that the message is not in a group and is not a segment of a logical message. MQMF_NONE is defined to aid program documentation. It is not intended that this flag be used with any other, but as its value is zero, such use cannot be detected.

The *MsgFlags* field is partitioned into subfields; for details see "Report options and message flags" on page 2206.

The initial value of this field is MQMF_NONE. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgId (MQBYTE24):

This is a byte string that is used to distinguish one message from another. Generally, no two messages should have the same message identifier, although this is not disallowed by the queue manager. The message identifier is a permanent property of the message, and persists across restarts of the queue manager. Because the message identifier is a byte string and not a character string, the message identifier is *not* converted between character sets when the message flows from one queue manager to another.

For the MQPUT and MQPUT1 calls, if MQMI_NONE or MQPMO_NEW_MSG_ID is specified by the application, the queue manager generates a unique message identifier [1] when the message is put, and places it in the message descriptor sent with the message. The queue manager also returns this message identifier in the message descriptor belonging to the sending application. The application can use this value to record information about particular messages, and to respond to queries from other parts of the application.

If the message is being put to a topic, the queue manager generates unique message identifiers as necessary for each message published. If MQPMO_NEW_MSG_ID is specified by the application, the queue manager generates a unique message identifier to return on output. If MQMI_NONE is specified by the application, the value of the *MsgId* field in the MQMD is unchanged on return from the call.

See the description of MQPMO_RETAIN in "MQPMO options (MQLONG)" on page 1797 for more details about retained publications.

If the message is being put to a distribution list, the queue manager generates unique message identifiers as necessary, but the value of the *MsgId* field in MQMD is unchanged on return from the call, even if MQMI_NONE or MQPMO_NEW_MSG_ID was specified. If the application needs to know the message identifiers generated by the queue manager, the application must provide MQPMR records containing the *MsgId* field.

The sending application can also specify a value for the message identifier other than MQMI_NONE; this stops the queue manager generating a unique message identifier. An application that is forwarding a message can use this to propagate the message identifier of the original message.

The queue manager does not use this field except to:

---

1. A *MsgId* generated by the queue manager consists of a 4-byte product identifier (AMQ¬ or CSQ¬ in either ASCII or EBCDIC, where ¬ represents a blank character), followed by a product-specific implementation of a unique string. In WebSphere MQ this contains the first 12 characters of the queue-manager name, and a value derived from the system clock. All queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order to ensure that message identifiers are unique. The ability to generate a unique string also depends on the system clock not being changed backward. To eliminate the possibility of a message identifier generated by the queue manager duplicating one generated by the application, the application must avoid generating identifiers with initial characters in the range A through I in ASCII or EBCDIC (X'41' through X'49' and X'C1' through X'C9'). However, the application is not prevented from generating identifiers with initial characters in these ranges.

- Generate a unique value if requested, as described above
- Deliver the value to the application that issues the get request for the message
- Copy the value to the *CorrelId* field of any report message that it generates about this message (depending on the *Report* options)

When the queue manager or a message channel agent generates a report message, it sets the *MsgId* field in the way specified by the *Report* field of the original message, either MQRO_NEW_MSG_ID or MQRO_PASS_MSG_ID. Applications that generate report messages must also do this.

For the MQGET call, *MsgId* is one of the five fields that can be used to retrieve a particular message from the queue. Normally the MQGET call returns the next message on the queue, but a particular message can be obtained by specifying one or more of the five selection criteria, in any combination; these fields are:

- *MsgId*
- *CorrelId*
- *GroupId*
- *MsgSeqNumber*
- *Offset*

The application sets one or more of these field to the values required, and then sets the corresponding MQMO_* match options in the *MatchOptions* field in MQGMO to use those fields as selection criteria. Only messages that have the specified values in those fields are candidates for retrieval. The default for the *MatchOptions* field (if not altered by the application) is to match both the message identifier and the correlation identifier.

On z/OS, the selection criteria that you can use are restricted by the type of index used for the queue. See the *IndexType* queue attribute for further details.

Normally, the message returned is the *first* message on the queue that satisfies the selection criteria. But if MQGMO_BROWSE_NEXT is specified, the message returned is the *next* message that satisfies the selection criteria; the scan for this message starts with the message *following* the current cursor position.

Note: The queue is scanned sequentially for a message that satisfies the selection criteria, so retrieval times are slower than if no selection criteria are specified, especially if many messages have to be scanned before a suitable one is found. The exceptions to this are:

- an MQGET call by *CorrelId* on 64-bit distributed platforms where the *CorrelId* index eliminates the need to perform a true sequential scan.
- an MQGET call by *IndexType* on z/OS.

In both these cases, retrieval performance is improved.

See Table 154 on page 1675 for more information about how selection criteria are used in various situations.

Specifying MQMI_NONE as the message identifier has the same effect as *not* specifying MQMO_MATCH_MSG_ID, that is, *any* message identifier matches.

This field is ignored if the MQGMO_MSG_UNDER_CURSOR option is specified in the *GetMsgOpts* parameter on the MQGET call.

On return from an MQGET call, the *MsgId* field is set to the message identifier of the message returned (if any).

The following special value can be used:

MQMI_NONE
No message identifier is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQMI_NONE_ARRAY is also defined; this has the same value as MQMI_NONE, but is an array of characters instead of a string.

This is an input/output field for the MQGET, MQPUT, and MQPUT1 calls. The length of this field is given by MQ_MSG_ID_LENGTH. The initial value of this field is MQMI_NONE.

MsgSeqNumber (MQLONG):

This is the sequence number of a logical message within a group.

Sequence numbers start at 1, and increase by 1 for each new logical message in the group, up to a maximum of 999 999 999. A physical message that is not in a group has a sequence number of 1.

The application does not have to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_MSG_SEQ_NUMBER is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application requires more control, or the call is MQPUT1, the application must ensure that *MsgSeqNumber* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Physical order on a queue. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

On input to the MQGET call, the queue manager uses the value shown in Table 154 on page 1675. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is one. This field is ignored if *Version* is less than MQMD_VERSION_2.

MsgType (MQLONG):

This indicates the type of the message. Message types are grouped as follows:

MQMT_SYSTEM_FIRST
Lowest value for system-defined message types.

MQMT_SYSTEM_LAST
Highest value for system-defined message types.

The following values are currently defined within the system range:

MQMT_DATAGRAM
The message is one that does not require a reply.

MQMT_REQUEST
The message is one that requires a reply.

Specify the name of the queue to which to send the reply in the *ReplyToQ* field. The *Report* field indicates how to set the *MsgId* and *CorrelId* of the reply.

MQMT_REPLY
The message is the reply to an earlier request message (MQMT_REQUEST). The message must be sent to the queue indicated by the *ReplyToQ* field of the request message. Use the *Report* field of the request to control how to set the *MsgId* and *CorrelId* of the reply.

Note: The queue manager does not enforce the request-reply relationship; this is an application responsibility.

MQMT_REPORT
The message is reporting on some expected or unexpected occurrence, usually related to some other message (for example, a request message was received that contained data that was not valid). Send the message to the queue indicated by the *ReplyToQ* field of the message descriptor of the original message. Set the *Feedback* field s to indicate the nature of the report. Use the *Report* field of the original message to control how to set the *MsgId* and *CorrelId* of the report message.

Report messages generated by the queue manager or message channel agent are always sent to the *ReplyToQ* queue, with the *Feedback* and *CorrelId* fields set as described above.

Application-defined values can also be used. They must be within the following range:

MQMT_APPL_FIRST
Lowest value for application-defined message types.

MQMT_APPL_LAST
Highest value for application-defined message types.

For the MQPUT and MQPUT1 calls, the *MsgType* value must be within either the system-defined range or the application-defined range; if it is not, the call fails with reason code MQRC_MSG_TYPE_ERROR.

This is an output field for the MQGET call, and an input field for MQPUT and MQPUT1 calls. The initial value of this field is MQMT_DATAGRAM.

Offset (MQLONG):

This is the offset in bytes of the data in the physical message from the start of the logical message of which the data forms part. This data is called a *segment*. The offset is in the range 0 through 999 999 999. A physical message that is not a segment of a logical message has an offset of zero.

The application does not need to set this field on the MQPUT or MQGET call if:

- On the MQPUT call, MQPMO_LOGICAL_ORDER is specified.
- On the MQGET call, MQMO_MATCH_OFFSET is *not* specified.

These are the recommended ways of using these calls for messages that are not report messages. However, if the application does not comply with these conditions, or the call is MQPUT1, the application must ensure that *Offset* is set to an appropriate value.

On input to the MQPUT and MQPUT1 calls, the queue manager uses the value described in Physical order on a queue. On output from the MQPUT and MQPUT1 calls, the queue manager sets this field to the value that was sent with the message.

For a report message reporting on a segment of a logical message, the *OriginalLength* field (provided it is not MQOL_UNDEFINED) is used to update the offset in the segment information retained by the queue manager.

On input to the MQGET call, the queue manager uses the value shown in Table 154 on page 1675. On output from the MQGET call, the queue manager sets this field to the value for the message retrieved.

The initial value of this field is zero. This field is ignored if *Version* is less than MQMD_VERSION_2.

OriginalLength (MQLONG):

This field is relevant only for report messages that are segments. It specifies the length of the message segment to which the report message relates; it does not specify the length of the logical message of which the segment forms part, or the length of the data in the report message.

Note: When generating a report message for a message that is a segment, the queue manager and message channel agent copy into the MQMD for the report message the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags*, fields from the original message. As a result, the report message is also a segment. Applications that generate report messages must do the same, and set the *OriginalLength* field correctly.

The following special value is defined:

MQOL_UNDEFINED
Original length of message not defined.

OriginalLength is an input field on the MQPUT and MQPUT1 calls, but the value that the application provides is accepted only in particular circumstances:

- If the message being put is a segment and is also a report message, the queue manager accepts the value specified. The value must be:
  - Greater than zero if the segment is not the last segment
  - Not less than zero if the segment is the last segment
  - Not less than the length of data present in the message

  If these conditions are not satisfied, the call fails with reason code MQRC_ORIGINAL_LENGTH_ERROR.
- If the message being put is a segment but not a report message, the queue manager ignores the field and uses the length of the application message data instead.
- In all other cases, the queue manager ignores the field and uses the value MQOL_UNDEFINED instead.

This is an output field on the MQGET call.

The initial value of this field is MQOL_UNDEFINED. This field is ignored if *Version* is less than MQMD_VERSION_2.

Persistence (MQLONG):

This indicates whether the message survives system failures and restarts of the queue manager. For the MQPUT and MQPUT1 calls, the value must be one of the following:

MQPER_PERSISTENT
The message survives system failures and restarts of the queue manager. Once the message has been put, and the unit of work in which it was put has been committed (if the message is put as part of a unit of work), the message is preserved on auxiliary storage. It remains there until the message is removed from the queue, and the unit of work in which it was got has been committed (if the message is retrieved as part of a unit of work).

When a persistent message is sent to a remote queue, a store-and-forward mechanism holds the message at each queue manager along the route to the destination, until the message is known to have arrived at the next queue manager.

Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues that map to a CFSTRUCT object at CFLEVEL(2) or below, or where the CFSTRUCT object is defined as RECOVER(NO).

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT
The message does not usually survive system failures or queue manager restarts. This applies even if an intact copy of the message is found on auxiliary storage when the queue manager restarts.

In the case of NPMCLASS (HIGH) queues nonpersistent messages survive a normal queue manager shutdown and restart.

In the case of shared queues, nonpersistent messages survive queue manager restarts in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

MQPER_PERSISTENCE_AS_Q_DEF

- If the queue is a cluster queue, the persistence of the message is taken from the *DefPersistence* attribute defined at the *destination* queue manager that owns the particular instance of the queue on which the message is placed. Usually, all instances of a cluster queue have the same value for the *DefPersistence* attribute, although this is not mandated.

  The value of *DefPersistence* is copied into the *Persistence* field when the message is placed on the destination queue. If *DefPersistence* is changed subsequently, messages that have already been placed on the queue are not affected.
- If the queue is not a cluster queue, the persistence of the message is taken from the *DefPersistence* attribute defined at the *local* queue manager, even if the destination queue manager is remote.

  If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path. This can be:
  - An alias queue
  - A local queue
  - A local definition of a remote queue
  - A queue-manager alias
  - A transmission queue (for example, the *DefXmitQName* queue)

  The value of *DefPersistence* is copied into the *Persistence* field when the message is put. If *DefPersistence* is changed subsequently, messages that have already been put are not affected.

Both persistent and nonpersistent messages can exist on the same queue.

When replying to a message, applications must use the persistence of the request message for the reply message.

For an MQGET call, the value returned is either MQPER_PERSISTENT or MQPER_NOT_PERSISTENT.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPER_PERSISTENCE_AS_Q_DEF.

Priority (MQLONG):

For the MQPUT and MQPUT1 calls, the value must be greater than or equal to zero; zero is the lowest priority. The following special value can also be used:

MQPRI_PRIORITY_AS_Q_DEF

- If the queue is a cluster queue, the priority for the message is taken from the *DefPriority* attribute as defined at the *destination* queue manager that owns the particular instance of the queue on which the message is placed. Usually, all instances of a cluster queue have the same value for the *DefPriority* attribute, although this is not mandated.

The value of *DefPriority* is copied into the *Priority* field when the message is placed on the destination queue. If *DefPriority* is changed subsequently, messages that have already been placed on the queue are not affected.

- If the queue is not a cluster queue, the priority for the message is taken from the *DefPriority* attribute as defined at the *local* queue manager, even if the destination queue manager is remote.

  If there is more than one definition in the queue-name resolution path, the default priority is taken from the value of this attribute in the *first* definition in the path. This can be:

  – An alias queue
  – A local queue
  – A local definition of a remote queue
  – A queue-manager alias
  – A transmission queue (for example, the *DefXmitQName* queue)

  The value of *DefPriority* is copied into the *Priority* field when the message is put. If *DefPriority* is changed subsequently, messages that have already been put are not affected.

The value returned by the MQGET call is always greater than or equal to zero; the value MQPRI_PRIORITY_AS_Q_DEF is never returned.

If a message is put with a priority greater than the maximum supported by the local queue manager (this maximum is given by the *MaxPriority* queue-manager attribute), the message is accepted by the queue manager, but placed on the queue at the queue manager's maximum priority; the MQPUT or MQPUT1 call completes with MQCC_WARNING and reason code MQRC_PRIORITY_EXCEEDS_MAXIMUM. However, the *Priority* field retains the value specified by the application that put the message.

On z/OS, if a message with a MsgSeqNumber of 1 is put to a queue that has a message delivery sequence of MQMDS_PRIORITY and an index type of MQIT_GROUP_ID, the queue might treat the message with a different priority. If the message was placed on the queue with a priority of 0 or 1, it is processed as though it has a priority of 2. This is because the order of messages placed on this type of queue is optimized to enable efficient group completeness tests. For more information on the message delivery sequence MQMDS_PRIORITY and the index type MQIT_GROUP_ID, see MsgDeliverySequence attribute.

When replying to a message, applications must use the priority of the request message for the reply message. In other situations, specifying MQPRI_PRIORITY_AS_Q_DEF allows priority tuning to be carried out without changing the application.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQPRI_PRIORITY_AS_Q_DEF.

PutApplName (MQCHAR28):

This is the name of application that put the message, and is part of the *origin context* of the message. The contents differ between platforms, and might differ between releases.

For more information about message context, see “Overview for MQMD” on page 1706 and Message context.

The format of *PutApplName* depends on the value of *PutApplType* and can change from one release to another. Changes are rare, but do happen if the environment changes.

When the queue manager sets this field (that is, for all options except MQPMO_SET_ALL_CONTEXT), it sets the field to a value that is determined by the environment:

- On z/OS, the queue manager uses:

- For z/OS batch, the 8-character job name from the JES JOB card
- For TSO, the 7-character TSO user identifier
- For CICS, the 8-character applid, followed by the 4-character tranid
- For IMS, the 8-character IMS system identifier, followed by the 8-character PSB name
- For XCF, the 8-character XCF group name, followed by the 16-character XCF member name
- For a message generated by a queue manager, the first 28 characters of the queue manager name
- For distributed queuing without CICS, the 8-character jobname of the channel initiator followed by the 8-character name of the module putting to the dead-letter queue followed by an 8-character task identifier.

The name or names are each padded to the right with blanks, as is any space in the remainder of the field. Where there is more than one name, there is no separator between them.

- On Windows systems, the queue manager uses:
  - For a CICS application, the CICS transaction name
  - For a non-CICS application, the rightmost 28 characters of the fully-qualified name of the executable
- On IBM i, the queue manager uses the fully-qualified job name.
- On UNIX systems, the queue manager uses:
  - For a CICS application, the CICS transaction name
  - For a non-CICS application, MQ asks the operating system for the name of the process. This is returned as the program file name, without full path. Then MQ places this process name in the MQMD.PutApplName field as follows:

    AIX If the name is less than or equal to 28 bytes, then the name is inserted, padded to the right with spaces.

    If the name is greater than 28 bytes, then the leftmost 28 bytes of the name are inserted.

    Linux and Solaris
    If the name is less than or equal to 15 bytes, then the name is inserted, padded to the right with spaces.

    If the name is greater than 15 bytes, then the leftmost 15 bytes of the name are inserted, padded to the right with spaces.

    HP-UX
    If the name is less than or equal to 14 bytes, then the name is inserted, padded to the right with spaces.

    If the name is greater than 14 bytes, then the leftmost 14 bytes of the name are inserted, padded to the right with spaces.

    For example, if you run `/opt/mqm/samp/bin/amqsput QNAME QMNAME`, then the PutApplName is `'amqsput                       '`. There are 21 space characters of padding in this MQCHAR28 field. Note that the full path including `/opt/mqm/samp/bin` is not included in the PutApplName.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The null character and any following characters are converted to blanks by the queue manager. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

PutApplType (MQLONG):

This is the type of application that put the message, and is part of the **origin context** of the message. For more information about message context, see "Overview for MQMD" on page 1706 and Message context.

PutApplType can have one of the following standard types. You can also define your own types, but only with values in the range MQAT_USER_FIRST through MQAT_USER_LAST.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_BROKER
Broker.

MQAT_CICS
CICS transaction.

MQAT_CICS_BRIDGE
CICS bridge.

MQAT_CICS_VSE
CICS/VSE transaction.

MQAT_DOS
WebSphere MQ MQI client application on PC DOS.

MQAT_DQM
Distributed queue manager agent.

MQAT_GUARDIAN
Tandem Guardian application (same value as MQAT_NSK).

MQAT_IMS
IMS application.

MQAT_IMS_BRIDGE
IMS bridge.

MQAT_JAVA
Java.

MQAT_MVS
MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT
Lotus Notes Agent application.

MQAT_NSK
HP Integrity NonStop Server application.

MQAT_OS390
OS/390 application (same value as MQAT_ZOS).

MQAT_OS400
IBM i application.

MQAT_QMGR
Queue manager.

MQAT_UNIX
UNIX application.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

MQAT_XCF
XCF.

MQAT_ZOS
z/OS application.

MQAT_DEFAULT
Default application type.

This is the default application type for the platform on which the application is running.

Note: The value of this constant is environment-specific. Because of this, always compile the application using the header, include, or COPY files that are appropriate to the platform on which the application will run.

MQAT_UNKNOWN
Use this value to indicate that the application type is unknown, even though other context information is present.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

The following special value can also occur:

MQAT_NO_CONTEXT
This value is set by the queue manager when a message is put with no context (that is, the MQPMO_NO_CONTEXT context option is specified).

When a message is retrieved, *PutApplType* can be tested for this value to decide whether the message has context (it is recommended that *PutApplType* is never set to MQAT_NO_CONTEXT, by an application using MQPMO_SET_ALL_CONTEXT, if any of the other context fields are nonblank).

When the queue manager generates this information as a result of an application put, the field is set to a value that is determined by the environment. On IBM i, it is set to MQAT_OS400; the queue manager never uses MQAT_CICS on IBM i.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The initial value of this field is MQAT_NO_CONTEXT.

PutDate (MQCHAR8):

This is the date when the message was put, and is part of the **origin context** of the message. For more information about message context, see "Overview for MQMD" on page 1706 and Message context.

The format used for the date when this field is generated by the queue manager is:

- YYYYMMDD

where the characters represent:

YYYY year (four numeric digits)

MM month of year (01 through 12)

DD day of month (01 through 31)

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

If the message was put as part of a unit of work, the date is that when the message was put, and not the date when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. The contents of the field are not checked by the queue manager, except that any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_DATE_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

PutTime (MQCHAR8):

This is the time when the message was put, and is part of the **origin context** of the message. For more information about message context, see "Overview for MQMD" on page 1706 and Message context.

The format used for the time when this field is generated by the queue manager is:

- HHMMSSTH

where the characters represent (in order):

HH hours (00 through 23)

MM minutes (00 through 59)

SS seconds (00 through 59; see note)

T tenths of a second (0 through 9)

H hundredths of a second (0 through 9)

Note: If the system clock is synchronized to a very accurate time standard, it is possible on rare occasions for 60 or 61 to be returned for the seconds in *PutTime*. This happens when leap seconds are inserted into the global time standard.

Greenwich Mean Time (GMT) is used for the *PutDate* and *PutTime* fields, subject to the system clock being set accurately to GMT.

If the message was put as part of a unit of work, the time is that when the message was put, and not the time when the unit of work was committed.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. The queue manager does not check the contents of the field, except that any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

This is an output field for the MQGET call. The length of this field is given by MQ_PUT_TIME_LENGTH. The initial value of this field is the null string in C, and 8 blank characters in other programming languages.

ReplyToQ (MQCHAR48):

This is the name of the message queue to which the application that issued the get request for the message sends MQMT_REPLY and MQMT_REPORT messages. The name is the local name of a queue that is defined on the queue manager identified by *ReplyToQMgr*. This queue must not be a model queue, although the sending queue manager does not verify this when the message is put.

For the MQPUT and MQPUT1 calls, this field must not be blank if the *MsgType* field has the value MQMT_REQUEST, or if any report messages are requested by the *Report* field. However, the value specified (or substituted) is passed on to the application that issues the get request for the message, whatever the message type.

If the *ReplyToQMgr* field is blank, the local queue manager looks up the *ReplyToQ* name in its own queue definitions. If a local definition of a remote queue exists with this name, the *ReplyToQ* value in the transmitted message is replaced by the value of the *RemoteQName* attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, *ReplyToQ* is unchanged.

If the name is specified, it can contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise no check is made that the name satisfies the naming rules for queues; this is also true for the name transmitted, if the *ReplyToQ* is replaced in the transmitted message. The only check made is that a name has been specified, if the circumstances require it.

If a reply-to queue is not required, set the *ReplyToQ* field to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; do not leave the field uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

If a message that requires a report message cannot be delivered, and the report message also cannot be delivered to the queue specified, both the original message and the report message go to the dead-letter (undelivered-message) queue (see the *DeadLetterQName* attribute described in "Attributes for the queue manager" on page 2096).

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ReplyToQMgr (MQCHAR48):

This is the name of the queue manager to which to send the reply message or report message. *ReplyToQ* is the local name of a queue that is defined on this queue manager.

If the *ReplyToQMgr* field is blank, the local queue manager looks up the *ReplyToQ* name in its queue definitions. If a local definition of a remote queue exists with this name, the *ReplyToQMgr* value in the transmitted message is replaced by the value of the *RemoteQMgrName* attribute from the definition of the remote queue, and this value is returned in the message descriptor when the receiving application issues an MQGET call for the message. If a local definition of a remote queue does not exist, the *ReplyToQMgr* that is transmitted with the message is the name of the local queue manager.

If the name is specified, it can contain trailing blanks; the first null character and characters following it are treated as blanks. Otherwise no check is made that the name satisfies the naming rules for queue managers, or that this name is known to the sending queue manager; this is also true for the name transmitted, if the *ReplyToQMgr* is replaced in the transmitted message.

If a reply-to queue is not required, set the *ReplyToQMgr* field to blanks, or (in the C programming language) to the null string, or to one or more blanks followed by a null character; do not leave the field uninitialized.

For the MQGET call, the queue manager always returns the name padded with blanks to the length of the field.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Report (MQLONG):

A report message is a message about another message, used to inform an application about expected or unexpected events that relate to the original message. The *Report* field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also (for both reports and replies) how the message and correlation identifiers in the report or reply message are to be set. Any or all (or none) of the following types of report message can be requested:

- Exception
- Expiration
- Confirm on arrival (COA)
- Confirm on delivery (COD)
- Positive action notification (PAN)
- Negative action notification (NAN)

If more than one type of report message is required, or other report options are needed, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

The application that receives the report message can determine the reason that the report was generated by examining the *Feedback* field in the MQMD; see the *Feedback* field for more details.

The use of report options when putting a message to a topic can cause zero, one, or many report messages to be generated and sent to the application. This is because the publication message may be sent to zero, one, or many subscribing applications.

Exception options: Specify one of the options listed to request an exception report message.

MQRO_EXCEPTION
A message channel agent generates this type of report when a message is sent to another queue manager and the message cannot be delivered to the specified destination queue. For example, the destination queue or an intermediate transmission queue might be full, or the message might be too big for the queue.

Generation of the exception report message depends on the persistence of the original message, and the speed of the message channel (normal or fast) through which the original message travels:

- For all persistent messages, and for nonpersistent messages traveling through normal message channels, the exception report is generated *only* if the action specified by the sending application for the error condition can be completed successfully. The sending application can specify one of the following actions to control the disposition of the original message when the error condition arises:
  - MQRO_DEAD_LETTER_Q (this places the original message on the dead-letter queue).
  - MQRO_DISCARD_MSG (this discards the original message).

  If the action specified by the sending application cannot be completed successfully, the original message is left on the transmission queue, and no exception report message is generated.
- For nonpersistent messages traveling through fast message channels, the original message is removed from the transmission queue and the exception report generated *even if* the specified action for the error condition cannot be completed successfully. For example, if MQRO_DEAD_LETTER_Q is specified, but the original message cannot be placed on the dead-letter queue because that queue is full, the exception report message is generated and the original message discarded.

  For more information about normal and fast message channels, see Nonpersistent message speed (NPMSPEED).

An exception report is not generated if the application that put the original message can be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call.

Applications can also send exception reports, to indicate that a message cannot be processed (for example, because it is a debit transaction that would cause the account to exceed its credit limit).

Message data from the original message is not included with the report message.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_DATA
This is the same as MQRO_EXCEPTION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_FULL_DATA
Exception reports with full data required.

This is the same as MQRO_EXCEPTION, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

Expiration options: Specify one of the options listed to request an expiration report message.

MQRO_EXPIRATION
This type of report is generated by the queue manager if the message is discarded before delivery to an application because its expiry time has passed (see the *Expiry* field). If this option is not set, no report message is generated if a message is discarded for this reason (even if you specify one of the MQRO_EXCEPTION_* options).

Message data from the original message is not included with the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_DATA
This is the same as MQRO_EXPIRATION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_FULL_DATA
This is the same as MQRO_EXPIRATION, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

Confirm-on-arrival options: Specify one of the options listed to request a confirm-on-arrival report message.

MQRO_COA
This type of report is generated by the queue manager that owns the destination queue when the message is placed on the destination queue. Message data from the original message is not included with the report message.

If the message is put as part of a unit of work, and the destination queue is a local queue, the COA report message generated by the queue manager can be retrieved only if the unit of work is committed.

A COA report is not generated if the *Format* field in the message descriptor is MQFMT_XMIT_Q_HEADER or MQFMT_DEAD_LETTER_HEADER. This prevents a COA report being generated if the message is put on a transmission queue, or is undeliverable and put on a dead-letter queue.

In the case of an IMS bridge queue, the COA report is generated when the message reaches the IMS queue (acknowledgment received from IMS) and not when the message is put in the MQ bridge queue. That means that if IMS is not active, no COA report is generated until IMS is started and a message is queued on the IMS queue.

The user that runs a program that puts a message with MQMD.Report=MQRO_COA must have +passid authority on the reply queue. If the user does not have +passid authority, the COA report message does not reach the reply queue. An attempt is made to put the report message on the dead letter queue.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_DATA
This is the same as MQRO_COA, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_FULL_DATA
This is the same as MQRO_COA, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

Confirm-on-delivery options: Specify one of the options listed to request a confirm-on-delivery report message.

MQRO_COD
This type of report is generated by the queue manager when an application retrieves the message from the destination queue in a way that deletes the message from the queue. Message data from the original message is not included with the report message.

If the message is retrieved as part of a unit of work, the report message is generated within the same unit of work, so that the report is not available until the unit of work is committed. If the unit of work is backed out, the report is not sent.

A COD report is not always generated if a message is retrieved with the MQGMO_MARK_SKIP_BACKOUT option. If the primary unit of work is backed out but the secondary unit of work is committed, the message is removed from the queue, but a COD report is not generated.

A COD report is not generated if the *Format* field in the message descriptor is MQFMT_DEAD_LETTER_HEADER. This prevents a COD report being generated if the message is undeliverable and put on a dead-letter queue.

MQRO_COD is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_DATA
This is the same as MQRO_COD, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

If MQGMO_ACCEPT_TRUNCATED_MSG is specified on the MQGET call for the original message, and the message retrieved is truncated, the amount of application message data placed in the report message depends on the environment:

- On z/OS, it is the minimum of:
  - The length of the original message
  - The length of the buffer used to retrieve the message
  - 100 bytes.
- In other environments, it is the minimum of:
  - The length of the original message
  - 100 bytes.

MQRO_COD_WITH_DATA is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_FULL_DATA
This is the same as MQRO_COD, except that all the application message data from the original message is included in the report message.

MQRO_COD_WITH_FULL_DATA is not valid if the destination queue is an XCF queue.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

Action-notification options: Specify one or both of the options listed to request that the receiving application send a positive-action or negative-action report message.

MQRO_PAN
This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has been performed successfully. The application generating the report determines whether any data is to be included with the report.

Other than conveying this request to the application retrieving the message, the queue manager takes no action based on this option. The retrieving application must generate the report if appropriate.

MQRO_NAN
This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has *not* been performed successfully. The application generating the report determines whether any data is to be included with the report. For example, you might want to include some data indicating why the request could not be performed.

Other than conveying this request to the application retrieving the message, the queue manager takes no action based on this option. The retrieving application must generate the report if appropriate.

The application must determine which conditions correspond to a positive action and which correspond to a negative action. However, if the request has been only partially performed, generate a NAN report rather than a PAN report if requested. Every possible condition must correspond to either a positive action, or a negative action, but not both.

Message-identifier options: Specify one of the options listed to control how the *MsgId* of the report message (or of the reply message) is to be set.

MQRO_NEW_MSG_ID
This is the default action, and indicates that if a report or reply is generated as a result of this message, a new *MsgId* is generated for the report or reply message.

MQRO_PASS_MSG_ID
If a report or reply is generated as a result of this message, the *MsgId* of this message is copied to the *MsgId* of the report or reply message.

The *MsgId* of a publication message will be different for each subscriber that receives a copy of the publication and therefore the *MsgId* copied into the report or reply message will be different for each one.

If this option is not specified, MQRO_NEW_MSG_ID is assumed.

Correlation-identifier options: Specify one of the options listed to control how the *CorrelId* of the report message (or of the reply message) is to be set.

MQRO_COPY_MSG_ID_TO_CORREL_ID
This is the default action, and indicates that if a report or reply is generated as a result of this message, the *MsgId* of this message is copied to the *CorrelId* of the report or reply message.

The *MsgId* of a publication message will be different for each subscriber that receives a copy of the publication and therefore the *MsgId* copied into the *CorrelId* of the report or reply message will be different for each one.

MQRO_PASS_CORREL_ID
If a report or reply is generated as a result of this message, the *CorrelId* of this message is copied to the *CorrelId* of the report or reply message.

The *CorrelId* of a publication message will be specific to a subscriber unless it uses the MQSO_SET_CORREL_ID option and sets the SubCorrelId field in the MQSD to MQCI_NONE. Therefore it is possible that the *CorrelId* copied into the *CorrelId* of the report or reply message will be different for each one.

If this option is not specified, MQRO_COPY_MSG_ID_TO_CORREL_ID is assumed.

Servers replying to requests or generating report messages must check whether the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options were set in the original message. If they were, the servers must take the action described for those options. If neither is set, the servers must take the corresponding default action.

Disposition options: Specify one of the options listed to control the disposition of the original message when it cannot be delivered to the destination queue. The application can set the disposition options independently of requesting exception reports.

MQRO_DEAD_LETTER_Q
This is the default action, and places the message on the dead-letter queue if the message cannot be delivered to the destination queue. This happens in the following situations:

- When the application that put the original message cannot be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call. An exception report message is generated, if one was requested by the sender.
- When the application that put the original message was putting to a topic

MQRO_DISCARD_MSG
This discards the message if it cannot be delivered to the destination queue. This happens in the following situations:

- When the application that put the original message cannot be notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call. An exception report message is generated, if one was requested by the sender.
- When the application that put the original message was putting to a topic

If you want to return the original message to the sender, without the original message being placed on the dead-letter queue, the sender must specify MQRO_DISCARD_MSG with MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_PASS_DISCARD_AND_EXPIRY
If this option is set on a message, and a report or reply is generated because of it, the message descriptor of the report inherits:

- MQRO_DISCARD_MSG if it was set.
- The remaining expiry time of the message (if this is not an expiry report). If this is an expiry report the expiry time is set to 60 seconds.

Activity option

MQRO_ACTIVITY
Using this value allows the route of **any** message to be traced throughout a queue manager network. The report option can be specified on any current user message, instantly allowing you to begin calculating the route of the message through the network.

If the application generating the message cannot switch on activity reports, reports can be turned on using an API crossing exit supplied by queue manager administrators.

Note:

1. The fewer the queue managers in the network that are able to generate activity reports, the less detailed the route.
2. The activity reports might be difficult to place in the correct order to determine the route taken.
3. The activity reports might not be able to find a route to their requested destination.
4. Messages with this report option set must be accepted by any queue manager, even if they do not understand the option. This allows the report option to be set on any user message, even if they are processed by a non Version 6.0 or later queue manager.
5. If a process, either a queue manager or a user process, performs an activity on a message with this option set it can choose to generate and put an activity report.

Default option: Specify the following if no report options are required:

MQRO_NONE
Use this value to indicate that no other options have been specified. MQRO_NONE is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

General information:

1. All report types required must be specifically requested by the application sending the original message. For example, if a COA report is requested but an exception report is not, a COA report is generated when the message is placed on the destination queue, but no exception report is generated if the destination queue is full when the message arrives there. If no *Report* options are set, no report messages are generated by the queue manager or message channel agent (MCA).

   Some report options can be specified even though the local queue manager does not recognize them; this is useful when the option is to be processed by the *destination* queue manager. See "Report options and message flags" on page 2206 for more details.

   If a report message is requested, the name of the queue to which to send the report must be specified in the *ReplyToQ* field. When a report message is received, the nature of the report can be determined by examining the *Feedback* field in the message descriptor.
2. If the queue manager or MCA that generates a report message cannot put the report message on the reply queue (for example, because the reply queue or transmission queue is full), the report message is placed instead on the dead-letter queue. If that *also* fails, or there is no dead-letter queue, the action taken depends on the type of the report message:
   - If the report message is an exception report, the message that generated the exception report is left on its transmission queue; this ensures that the message is not lost.
   - For all other report types, the report message is discarded and processing continues normally. This is done because either the original message has already been delivered safely (for COA or COD report messages), or is no longer of any interest (for an expiration report message).

   Once a report message has been placed successfully on a queue (either the destination queue or an intermediate transmission queue), the message is no longer subject to special processing; it is treated just like any other message.
3. When the report is generated, the *ReplyToQ* queue is opened and the report message put using the authority of the *UserIdentifier* in the MQMD of the message causing the report, except in the following cases:
   - Exception reports generated by a receiving MCA are put with whatever authority the MCA used when it tried to put the message causing the report.
   - COA reports generated by the queue manager are put with whatever authority was used when the message causing the report was put on the queue manager generating the report. For example, if the message was put by a receiving MCA using the MCA's user identifier, the queue manager puts the COA report using the MCA's user identifier.

Applications generating reports must use the same authority as they use to generate a reply; this is usually the authority of the user identifier in the original message.

If the report has to travel to a remote destination, senders and receivers can decide whether to accept it, in the same way as they do for other messages.

4. If a report message with data is requested:
   - The report message is always generated with the amount of data requested by the sender of the original message. If the report message is too big for the reply queue, the processing described above occurs; the report message is never truncated to fit on the reply queue.
   - If the *Format* of the original message is MQFMT_XMIT_Q_HEADER, the data included in the report does not include the MQXQH. The report data starts with the first byte of the data beyond the MQXQH in the original message. This occurs whether or not the queue is a transmission queue.
5. If a COA, COD, or expiration report message is received at the reply queue, it is guaranteed that the original message arrived, was delivered, or expired, as appropriate. However, if one or more of these report messages is requested and is *not* received, the reverse cannot be assumed, because one of the following might have occurred:
   a. The report message is held up because a link is down.
   b. The report message is held up because a blocking condition exists at an intermediate transmission queue or at the reply queue (for example, the queue is full or inhibited for puts).
   c. The report message is on a dead-letter queue.
   d. When the queue manager was attempting to generate the report message, it could neither put it on the appropriate queue, nor on the dead-letter queue, so the report message could not be generated.
   e. A failure of the queue manager occurred between the action being reported (arrival, delivery, or expiry), and generation of the corresponding report message. (This does not happen for COD report messages if the application retrieves the original message within a unit of work, as the COD report message is generated within the same unit of work.)

   Exception report messages can be held up in the same way for reasons 1, 2, and 3 above. However, when an MCA cannot generate an exception report message (the report message cannot be put either on the reply queue or the dead-letter queue), the original message remains on the transmission queue at the sender, and the channel is closed. This occurs irrespective of whether the report message was to be generated at the sending or the receiving end of the channel.
6. If the original message is temporarily blocked (resulting in an exception report message being generated and the original message being put on a dead-letter queue), but the blockage clears and an application then reads the original message from the dead-letter queue and puts it again to its destination, the following might occur:
   - Even though an exception report message has been generated, the original message eventually arrives successfully at its destination.
   - More than one exception report message is generated in respect of a single original message, because the original message might encounter another blockage later.

Report messages when putting to a topic:

1. Reports can be generated when putting a message to a topic. This message will be sent to all subscribers to the topic, which could be zero, one, or many. This should be taken into account when choosing to use report options as many report messages could be generated as a result.
2. When putting a message to a topic, there may be many destination queues that are to be given a copy of the message. If some of these destination queues have a problem, such as queue full, then the successful completion of the MQPUT depends on the setting of NPMSGDLV or PMSGDLV (depending on the persistence of the message). If the setting is such that message delivery to the destination queue must be successful (for example, it is a persistent message to a durable subscriber and PMSGDLV is set to ALL or ALLDUR), then success is defined as one of the following criteria being met:
   - Successful put to the subscriber queue

- Use of MQRO_DEAD_LETTER_Q and a successful put to the Dead-letter queue if the subscriber queue cannot take the message
- Use of MQRO_DISCARD_MSG if the subscriber queue cannot take the message.

Report messages for message segments:

1. Report messages can be requested for messages that have segmentation allowed (see the description of the MQMF_SEGMENTATION_ALLOWED flag). If the queue manager finds it necessary to segment the message, a report message can be generated for each of the segments that subsequently encounters the relevant condition. Applications must be prepared to receive multiple report messages for each type of report message requested. Use the *GroupId* field in the report message to correlate the multiple reports with the group identifier of the original message, and the *Feedback* field identify the type of each report message.
2. If MQGMO_LOGICAL_ORDER is used to retrieve report messages for segments, be aware that reports of *different types* might be returned by the successive MQGET calls. For example, if both COA and COD reports are requested for a message that is segmented by the queue manager, the MQGET calls for the report messages might return the COA and COD report messages interleaved in an unpredictable fashion. Avoid this by using the MQGMO_COMPLETE_MSG option (optionally with MQGMO_ACCEPT_TRUNCATED_MSG). MQGMO_COMPLETE_MSG causes the queue manager to reassemble report messages that have the same report type. For example, the first MQGET call might reassemble all the COA messages relating to the original message, and the second MQGET call might reassemble all the COD messages. Which is reassembled first depends on which type of report message occurs first on the queue.
3. Applications that themselves put segments can specify different report options for each segment. However, note the following points:
   - If the segments are retrieved using the MQGMO_COMPLETE_MSG option, only the report options in the *first* segment are honored by the queue manager.
   - If the segments are retrieved one by one, and most of them have one of the MQRO_COD_* options, but at least one segment does not, you cannot use the MQGMO_COMPLETE_MSG option to retrieve the report messages with a single MQGET call, or use the MQGMO_ALL_SEGMENTS_AVAILABLE option to detect when all the report messages have arrived.
4. In an MQ network, the queue managers can have different capabilities. If a report message for a segment is generated by a queue manager or MCA that does not support segmentation, the queue manager or MCA does not by default include the necessary segment information in the report message, and this might make it difficult to identify the original message that caused the report to be generated. Avoid this difficulty by requesting data with the report message, that is, by specifying the appropriate MQRO_*_WITH_DATA or MQRO_*_WITH_FULL_DATA options. However, be aware that if MQRO_*_WITH_DATA is specified, *less than* 100 bytes of application message data might be returned to the application that retrieves the report message, if the report message is generated by a queue manager or MCA that does not support segmentation.

Contents of the message descriptor for a report message: When the queue manager or message channel agent (MCA) generates a report message, it sets the fields in the message descriptor to the following values, and then puts the message in the normal way.

Field in MQMD	Value used
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_2
Report	MQRO_NONE
MsgType	MQMT_REPORT
Expiry	MQEI_UNLIMITED
Feedback	As appropriate for the nature of the report (MQFB_COA, MQFB_COD, MQFB_EXPIRATION, or an MQRC_* value)
Encoding	Copied from the original message descriptor
CodedCharSetId	Copied from the original message descriptor
Format	Copied from the original message descriptor
Priority	Copied from the original message descriptor
Persistence	Copied from the original message descriptor
MsgId	As specified by the report options in the original message descriptor
CorrelId	As specified by the report options in the original message descriptor
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Name of queue manager
UserIdentifier	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
AccountingToken	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
ApplIdentityData	As set by the MQPMO_PASS_IDENTITY_CONTEXT option
PutApplType	MQAT_QMGR, or as appropriate for the message channel agent
PutApplName	First 28 bytes of the queue-manager name or message channel agent name. For report messages generated by the IMS bridge, this field contains the XCF group name and XCF member name of the IMS system to which the message relates.
PutDate	Date when report message is sent
PutTime	Time when report message is sent
ApplOriginData	Blanks
GroupId	Copied from the original message descriptor
MsgSeqNumber	Copied from the original message descriptor
Offset	Copied from the original message descriptor
MsgFlags	Copied from the original message descriptor
OriginalLength	Copied from the original message descriptor if not MQOL_UNDEFINED, and set to the length of the original message data otherwise

An application generating a report is recommended to set similar values, except for the following:

- The *ReplyToQMgr* field can be set to blanks (the queue manager changes this to the name of the local queue manager when the message is put).
- Set the context fields using the option that would have been used for a reply, normally MQPMO_PASS_IDENTITY_CONTEXT.

Analyzing the report field: The *Report* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report must use one of the techniques described in "Analyzing the report field" on page 2208.

This is an output field for the MQGET call, and an input field for the MQPUT and MQPUT1 calls. The initial value of this field is MQRO_NONE.

StrucId (MQCHAR4):

This is the structure identifier, and must be:

MQMD_STRUC_ID
Identifier for message descriptor structure.

For the C programming language, the constant MQMD_STRUC_ID_ARRAY is also defined; this has the same value as MQMD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQMD_STRUC_ID.

UserIdentifier (MQCHAR12):

This is part of the **identity context** of the message. For more information about message context, see "Overview for MQMD" on page 1706 and Message context.

UserIdentifier specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it.

After a message has been received, use *UserIdentifier* in the *AlternateUserId* field of the *ObjDesc* parameter of a subsequent MQOPEN or MQPUT1 call to perform the authorization check for the *UserIdentifier* user instead of the application performing the open.

When the queue manager generates this information for an MQPUT or MQPUT1 call:

- On z/OS, the queue manager uses the *AlternateUserId* from the *ObjDesc* parameter of the MQOPEN or MQPUT1 call if the MQOO_ALTERNATE_USER_AUTHORITY or MQPMO_ALTERNATE_USER_AUTHORITY option was specified. If the relevant option was not specified, the queue manager uses a user identifier determined from the environment.
- In other environments, the queue manager always uses a user identifier determined from the environment.

When the user identifier is determined from the environment:

- On z/OS, the queue manager uses:
  - For MVS (batch), the user identifier from the JES JOB card or started task
  - For TSO, the user identifier propagated to the job during job submission
  - For CICS, the user identifier associated with the task
  - For IMS, the user identifier depends on the type of application:
    - For:
      - Nonmessage BMP regions
      - Nonmessage IFP regions
      - Message BMP and message IFP regions that have *not* issued a successful GU call

      the queue manager uses the user identifier from the region JES JOB card or the TSO user identifier. If these are blank or null, it uses the name of the program specification block (PSB).
    - For:
      - Message BMP and message IFP regions that *have* issued a successful GU call
      - MPP regions

      the queue manager uses one of:
      - The signed-on user identifier associated with the message
      - The logical terminal (LTERM) name
      - The user identifier from the region JES JOB card
      - The TSO user identifier

    - The PSB name
- On IBM i, the queue manager uses the name of the user profile associated with the application job.
- On UNIX systems, the queue manager uses:
  - The application's logon name
  - The effective user identifier of the process if no logon is available
  - The user identifier associated with the transaction, if the application is a CICS transaction
- On Windows systems, the queue manager uses the first 12 characters of the logged-on user name.

This field is normally an output field generated by the queue manager but for an MQPUT or MQPUT1 call you can make this field an input/output field and specify the UserIdentification field instead of letting the queue manager generate this information. Specify either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT in the PutMsgOpts parameter and specify a user ID in the UserIdentifier field if you do not want the queue manager to generate the UserIdentifier field for an MQPUT or MQPUT1 call.

For the MQPUT and MQPUT1 calls, this is an input/output field if MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is specified in the *PutMsgOpts* parameter. Any information following a null character within the field is discarded. The queue manager converts the null character and any following characters to blanks. If MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT is not specified, this field is ignored on input and is an output-only field.

After the successful completion of an MQPUT or MQPUT1 call, this field contains the *UserIdentifier* that was transmitted with the message if it was put to a queue. This will be the value of *UserIdentifier* that is kept with the message if it is retained (see description of MQPMO_RETAIN for more details about retained publications) but is not used as the *UserIdentifier* when the message is sent as a publication to subscribers because they provide a value to override *UserIdentifier* in all publications sent to them. If the message has no context, the field is entirely blank.

This is an output field for the MQGET call. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

Version (MQLONG):

This is the structure version number, and must be one of the following:

MQMD_VERSION_1
Version-1 message descriptor structure.

This version is supported in all environments.

MQMD_VERSION_2
Version-2 message descriptor structure.

This version is supported in all WebSphere MQ V6.0 and later environments, plus WebSphere MQ MQI clients connected to these systems.

Note: When a version-2 MQMD is used, the queue manager performs additional checks on any MQ header structures that might be present at the beginning of the application message data; for further details see the usage notes for the MQPUT call.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQMD_CURRENT_VERSION
Current version of message descriptor structure.

This is always an input field. The initial value of this field is MQMD_VERSION_1.

Initial values and language declarations for MQMD:

Table 163. Initial values of fields in MQMD for MQMD

Field name	Name of constant	Value of constant
StrucId	MQMD_STRUC_ID	'MD'
Version	MQMD_VERSION_1	1
Report	MQRO_NONE	0
MsgType	MQMT_DATAGRAM	8
Expiry	MQEI_UNLIMITED	-1
Feedback	MQFB_NONE	0
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_Q_MGR	0
Format	MQFMT_NONE	Blanks
Priority	MQPRI_PRIORITY_AS_Q_DEF	-1
Persistence	MQPER_PERSISTENCE_AS_Q_DEF	2
MsgId	MQMI_NONE	Nulls
CorrelId	MQCI_NONE	Nulls
BackoutCount	None	0
ReplyToQ	None	Null string or blanks
ReplyToQMgr	None	Null string or blanks
UserIdentifier	None	Null string or blanks
AccountingToken	MQACT_NONE	Nulls
ApplIdentityData	None	Null string or blanks
PutApplType	MQAT_NO_CONTEXT	0
PutApplName	None	Null string or blanks
PutDate	None	Null string or blanks
PutTime	None	Null string or blanks
ApplOriginData	None	Null string or blanks
GroupId	MQGI_NONE	Nulls
MsgSeqNumber	None	1
Offset	None	0
MsgFlags	MQMF_NONE	0
OriginalLength	MQOL_UNDEFINED	-1

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQMD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQMD MyMD = {MQMD_DEFAULT};
```

C declaration:

```
typedef struct tagMQMD MQMD;
struct tagMQMD {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    Report;            /* Options for report messages */
  MQLONG    MsgType;           /* Message type */
  MQLONG    Expiry;            /* Message lifetime */
  MQLONG    Feedback;          /* Feedback or reason code */
  MQLONG    Encoding;          /* Numeric encoding of message data */
  MQLONG    CodedCharSetId;    /* Character set identifier of message
                                  data */
  MQCHAR8   Format;            /* Format name of message data */
  MQLONG    Priority;          /* Message priority */
  MQLONG    Persistence;       /* Message persistence */
  MQBYTE24  MsgId;             /* Message identifier */
  MQBYTE24  CorrelId;          /* Correlation identifier */
  MQLONG    BackoutCount;      /* Backout counter */
  MQCHAR48  ReplyToQ;          /* Name of reply queue */
  MQCHAR48  ReplyToQMgr;       /* Name of reply queue manager */
  MQCHAR12  UserIdentifier;    /* User identifier */
  MQBYTE32  AccountingToken;   /* Accounting token */
  MQCHAR32  ApplIdentityData;  /* Application data relating to
                                  identity */
  MQLONG    PutApplType;       /* Type of application that put the
                                  message */
  MQCHAR28  PutApplName;       /* Name of application that put the
                                  message */
  MQCHAR8   PutDate;           /* Date when message was put */
  MQCHAR8   PutTime;           /* Time when message was put */
  MQCHAR4   ApplOriginData;    /* Application data relating to origin */
  MQBYTE24  GroupId;           /* Group identifier */
  MQLONG    MsgSeqNumber;      /* Sequence number of logical message
                                  within group */
  MQLONG    Offset;            /* Offset of data in physical message
                                  from start of logical message */
  MQLONG    MsgFlags;          /* Message flags */
  MQLONG    OriginalLength;    /* Length of original message */
};
```

COBOL declaration:

```
**   MQMD structure
  10 MQMD.
**    Structure identifier
   15 MQMD-STRUCID          PIC X(4).
**    Structure version number
   15 MQMD-VERSION          PIC S9(9) BINARY.
**    Options for report messages
   15 MQMD-REPORT           PIC S9(9) BINARY.
**    Message type
   15 MQMD-MSGTYPE          PIC S9(9) BINARY.
**    Message lifetime
   15 MQMD-EXPIRY           PIC S9(9) BINARY.
**    Feedback or reason code
   15 MQMD-FEEDBACK         PIC S9(9) BINARY.
**    Numeric encoding of message data
   15 MQMD-ENCODING         PIC S9(9) BINARY.
**    Character set identifier of message data
   15 MQMD-CODEDCHARSETID   PIC S9(9) BINARY.
**    Format name of message data
   15 MQMD-FORMAT           PIC X(8).
**    Message priority
   15 MQMD-PRIORITY         PIC S9(9) BINARY.
**    Message persistence
   15 MQMD-PERSISTENCE      PIC S9(9) BINARY.
**    Message identifier
```

```
   15 MQMD-MSGID            PIC X(24).
**    Correlation identifier
   15 MQMD-CORRELID         PIC X(24).
**    Backout counter
   15 MQMD-BACKOUTCOUNT     PIC S9(9) BINARY.
**    Name of reply queue
   15 MQMD-REPLYTOQ         PIC X(48).
**    Name of reply queue manager
   15 MQMD-REPLYTOQMGR      PIC X(48).
**    User identifier
   15 MQMD-USERIDENTIFIER   PIC X(12).
**    Accounting token
   15 MQMD-ACCOUNTINGTOKEN  PIC X(32).
**    Application data relating to identity
   15 MQMD-APPLIDENTITYDATA PIC X(32).
**    Type of application that put the message
   15 MQMD-PUTAPPLTYPE      PIC S9(9) BINARY.
**    Name of application that put the message
   15 MQMD-PUTAPPLNAME      PIC X(28).
**    Date when message was put
   15 MQMD-PUTDATE          PIC X(8).
**    Time when message was put
   15 MQMD-PUTTIME          PIC X(8).
**    Application data relating to origin
   15 MQMD-APPLORIGINDATA   PIC X(4).
**    Group identifier
   15 MQMD-GROUPID          PIC X(24).
**    Sequence number of logical message within group
   15 MQMD-MSGSEQNUMBER     PIC S9(9) BINARY.
**    Offset of data in physical message from start of logical message
   15 MQMD-OFFSET           PIC S9(9) BINARY.
**    Message flags
   15 MQMD-MSGFLAGS         PIC S9(9) BINARY.
**    Length of original message
   15 MQMD-ORIGINALLENGTH   PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQMD based,
  3 StrucId          char(4),       /* Structure identifier */
  3 Version          fixed bin(31), /* Structure version number */
  3 Report           fixed bin(31), /* Options for report messages */
  3 MsgType          fixed bin(31), /* Message type */
  3 Expiry           fixed bin(31), /* Message lifetime */
  3 Feedback         fixed bin(31), /* Feedback or reason code */
  3 Encoding         fixed bin(31), /* Numeric encoding of message
                                       data */
  3 CodedCharSetId   fixed bin(31), /* Character set identifier of
                                       message data */
  3 Format           char(8),       /* Format name of message data */
  3 Priority         fixed bin(31), /* Message priority */
  3 Persistence      fixed bin(31), /* Message persistence */
  3 MsgId            char(24),      /* Message identifier */
  3 CorrelId         char(24),      /* Correlation identifier */
  3 BackoutCount     fixed bin(31), /* Backout counter */
  3 ReplyToQ         char(48),      /* Name of reply queue */
  3 ReplyToQMgr      char(48),      /* Name of reply queue manager */
  3 UserIdentifier   char(12),      /* User identifier */
  3 AccountingToken  char(32),      /* Accounting token */
  3 ApplIdentityData char(32),      /* Application data relating to
                                       identity */
  3 PutApplType      fixed bin(31), /* Type of application that put the
                                       message */
  3 PutApplName      char(28),      /* Name of application that put the
                                       message */
  3 PutDate          char(8),       /* Date when message was put */
```

```
  3 PutTime          char(8),       /* Time when message was put */
  3 ApplOriginData   char(4),       /* Application data relating to
                                       origin */
  3 GroupId          char(24),      /* Group identifier */
  3 MsgSeqNumber     fixed bin(31), /* Sequence number of logical
                                       message within group */
  3 Offset           fixed bin(31), /* Offset of data in physical
                                       message from start of logical
                                       message */
  3 MsgFlags         fixed bin(31), /* Message flags */
  3 OriginalLength   fixed bin(31); /* Length of original message */
```

High Level Assembler declaration:

```
MQMD                  DSECT
MQMD_STRUCID          DS   CL4   Structure identifier
MQMD_VERSION          DS   F     Structure version number
MQMD_REPORT           DS   F     Options for report messages
MQMD_MSGTYPE          DS   F     Message type
MQMD_EXPIRY           DS   F     Message lifetime
MQMD_FEEDBACK         DS   F     Feedback or reason code
MQMD_ENCODING         DS   F     Numeric encoding of message data
MQMD_CODEDCHARSETID   DS   F     Character set identifier of message
*                                data
MQMD_FORMAT           DS   CL8   Format name of message data
MQMD_PRIORITY         DS   F     Message priority
MQMD_PERSISTENCE      DS   F     Message persistence
MQMD_MSGID            DS   XL24  Message identifier
MQMD_CORRELID         DS   XL24  Correlation identifier
MQMD_BACKOUTCOUNT     DS   F     Backout counter
MQMD_REPLYTOQ         DS   CL48  Name of reply queue
MQMD_REPLYTOQMGR      DS   CL48  Name of reply queue manager
MQMD_USERIDENTIFIER   DS   CL12  User identifier
MQMD_ACCOUNTINGTOKEN  DS   XL32  Accounting token
MQMD_APPLIDENTITYDATA DS   CL32  Application data relating to identity
MQMD_PUTAPPLTYPE      DS   F     Type of application that put the
*                                message
MQMD_PUTAPPLNAME      DS   CL28  Name of application that put the
*                                message
MQMD_PUTDATE          DS   CL8   Date when message was put
MQMD_PUTTIME          DS   CL8   Time when message was put
MQMD_APPLORIGINDATA   DS   CL4   Application data relating to origin
MQMD_GROUPID          DS   XL24  Group identifier
MQMD_MSGSEQNUMBER     DS   F     Sequence number of logical message
*                                within group
MQMD_OFFSET           DS   F     Offset of data in physical message
*                                from start of logical message
MQMD_MSGFLAGS         DS   F     Message flags
MQMD_ORIGINALLENGTH   DS   F     Length of original message
*
MQMD_LENGTH           EQU  *-MQMD
                      ORG  MQMD
MQMD_AREA             DS   CL(MQMD_LENGTH)
```

Visual Basic declaration:

```
Type MQMD
  StrucId          As String*4  'Structure identifier'
  Version          As Long      'Structure version number'
  Report           As Long      'Options for report messages'
  MsgType          As Long      'Message type'
  Expiry           As Long      'Message lifetime'
  Feedback         As Long      'Feedback or reason code'
  Encoding         As Long      'Numeric encoding of message data'
  CodedCharSetId   As Long      'Character set identifier of message'
                                'data'
  Format           As String*8  'Format name of message data'
  Priority         As Long      'Message priority'
  Persistence      As Long      'Message persistence'
  MsgId            As MQBYTE24  'Message identifier'
  CorrelId         As MQBYTE24  'Correlation identifier'
  BackoutCount     As Long      'Backout counter'
  ReplyToQ         As String*48 'Name of reply queue'
  ReplyToQMgr      As String*48 'Name of reply queue manager'
  UserIdentifier   As String*12 'User identifier'
  AccountingToken  As MQBYTE32  'Accounting token'
  ApplIdentityData As String*32 'Application data relating to identity'
  PutApplType      As Long      'Type of application that put the'
                                'message'
  PutApplName      As String*28 'Name of application that put the'
                                'message'
  PutDate          As String*8  'Date when message was put'
  PutTime          As String*8  'Time when message was put'
  ApplOriginData   As String*4  'Application data relating to origin'
  GroupId          As MQBYTE24  'Group identifier'
  MsgSeqNumber     As Long      'Sequence number of logical message'
                                'within group'
  Offset           As Long      'Offset of data in physical message'
                                'from start of logical message'
  MsgFlags         As Long      'Message flags'
  OriginalLength   As Long      'Length of original message'
End Type
```

MQMDE - Message descriptor extension:

The following table summarizes the fields in the structure.

Table 164. Fields in MQMDE

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQMDE structure	StrucLength
Encoding	Numeric encoding of data that follows MQMDE	Encoding
CodedCharSetId	Character set identifier of data that follows MQMDE	CodedCharSetId
Format	Format name of data that follows MQMDE	Format
Flags	General flags	Flags
GroupId	Group identifier	GroupId
MsgSeqNumber	Sequence number of logical message within group	MsgSeqNumber
Offset	Offset of data in physical message from start of logical message	Offset
MsgFlags	Message flags	MsgFlags
OriginalLength	Length of original message	OriginalLength

Overview for MQMDE:

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQMDE structure describes the data that sometimes occurs preceding the application message data. The structure contains those MQMD fields that exist in the version-2 MQMD, but not in the version-1 MQMD.

Format name: MQFMT_MD_EXTENSION.

Character set and encoding: Data in MQMDE must be in the character set and encoding of the local queue manager; these are given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE for the C programming language.

Set the character set and encoding of the MQMDE into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQMDE structure is at the start of the message data), or
- The header structure that precedes the MQMDE structure (all other cases).

If the MQMDE is not in the queue manager's character set and encoding, the MQMDE is accepted but not honored, that is, the MQMDE is treated as message data.

Note: On Windows, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the queue-manager's encoding. Although numeric fields in the MQMD structure on the MQPUT, MQPUT1, and MQGET calls must be in the Micro Focus COBOL encoding, numeric fields in the MQMDE structure must be in the queue-manager's encoding. This latter is given by MQENC_NATIVE for the C programming language, and has the value 546.

Usage: Applications that use a version-2 MQMD will not encounter an MQMDE structure. However, specialized applications, and applications that continue to use a version-1 MQMD, might encounter an MQMDE in some situations. The MQMDE structure can occur in the following circumstances:

- Specified on the MQPUT and MQPUT1 calls
- Returned by the MQGET call
- In messages on transmission queues

MQMDE specified on MQPUT and MQPUT1 calls: On the MQPUT and MQPUT1 calls, if the application provides a version-1 MQMD, the application can optionally prefix the message data with an MQMDE, setting the *Format* field in MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. If the application does not provide an MQMDE, the queue manager assumes default values for the fields in the MQMDE. The default values that the queue manager uses are the same as the initial values for the structure; see Table 166 on page 1763.

If the application provides a version-2 MQMD *and* prefixes the application message data with an MQMDE, the structures are processed as shown in Table 165 on page 1760.

Table 165. Queue-manager action when MQMDE specified on MQPUT or MQPUT1 for MQMDE

MQMD version	Values of version-2 fields	Values of corresponding fields in MQMDE	Action taken by queue manager
1	-	Valid	MQMDE is honored
2	Default	Valid	MQMDE is honored
2	Not default	Valid	MQMDE is treated as message data
1 or 2	Any	Not valid	Call fails with an appropriate reason code
1 or 2	Any	MQMDE is in the wrong character set or encoding, or is an unsupported version	MQMDE is treated as message data
Note: On z/OS, if the application specifies a version-1 MQMD with an MQMDE, the queue manager validates the MQMDE only if the queue has an *IndexType* of MQIT_GROUP_ID.			

There is one special case. If the application uses a version-2 MQMD to put a message that is a segment (that is, the MQMF_SEGMENT or MQMF_LAST_SEGMENT flag is set), and the format name in the MQMD is MQFMT_DEAD_LETTER_HEADER, the queue manager generates an MQMDE structure and inserts it *between* the MQDLH structure and the data that follows it. In the MQMD that the queue manager retains with the message, the version-2 fields are set to their default values.

Several of the fields that exist in the version-2 MQMD but not the version-1 MQMD are input/output fields on MQPUT and MQPUT1. However, the queue manager does *not* return any values in the equivalent fields in the MQMDE on output from the MQPUT and MQPUT1 calls; if the application requires those output values, it must use a version-2 MQMD.

MQMDE returned by MQGET call: On the MQGET call, if the application provides a version-1 MQMD, the queue manager prefixes the message returned with an MQMDE, but only if one or more of the fields in the MQMDE has a nondefault value. The queue manager sets the *Format* field in MQMD to the value MQFMT_MD_EXTENSION to indicate that an MQMDE is present.

If the application provides an MQMDE at the start of the *Buffer* parameter, the MQMDE is ignored. On return from the MQGET call, it is replaced by the MQMDE for the message (if one is needed), or overwritten by the application message data (if the MQMDE is not needed).

If the MQGET call returns an MQMDE, the data in the MQMDE is usually in the queue manager's character set and encoding. However the MQMDE might be in some other character set and encoding if:

- The MQMDE was treated as data on the MQPUT or MQPUT1 call (see Table 165 for the circumstances that can cause this).
- The message was received from a remote queue manager connected by a TCP connection, and the receiving message channel agent (MCA) was not set up correctly.

Note: On Windows, applications compiled with Micro Focus COBOL use a value of MQENC_NATIVE that is different from the queue-manager's encoding (see above).

MQMDE in messages on transmission queues: Messages on transmission queues are prefixed with the MQXQH structure, which contains within it a version-1 MQMD. An MQMDE might also be present, positioned between the MQXQH structure and application message data, but it is usually present only if one or more of the fields in the MQMDE has a nondefault value.

Other MQ header structures can also occur between the MQXQH structure and the application message data. For example, when the dead-letter header MQDLH is present, and the message is not a segment, the order is:

- MQXQH (containing a version-1 MQMD)
- MQMDE
- MQDLH
- application message data

Fields for MQMDE:

The MQMDE structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the data that follows the MQMDE structure; it does not apply to character data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. The following special value can be used:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This specifies the numeric encoding of the data that follows the MQMDE structure; it does not apply to numeric data in the MQMDE structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that the field is valid. See the *Encoding* field described in "MQMD - Message descriptor" on page 1705 for more information about data encodings.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG):

The following flag can be specified:

MQMDEF_NONE
No flags.

The initial value of this field is MQMDEF_NONE.

Format (MQCHAR8):

This specifies the format name of the data that follows the MQMDE structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The queue manager does not check that this field is valid. See the *Format* field described in "MQMD - Message descriptor" on page 1705 for more information about format names.

The initial value of this field is MQFMT_NONE.

GroupId (MQBYTE24):

See the *GroupId* field described in "MQMD - Message descriptor" on page 1705. The initial value of this field is MQGI_NONE.

MsgFlags (MQLONG):

See the *MsgFlags* field described in "MQMD - Message descriptor" on page 1705. The initial value of this field is MQMF_NONE.

MsgSeqNumber (MQLONG):

See the *MsgSeqNumber* field described in "MQMD - Message descriptor" on page 1705. The initial value of this field is 1.

Offset (MQLONG):

See the *Offset* field described in "MQMD - Message descriptor" on page 1705. The initial value of this field is 0.

OriginalLength (MQLONG):

See the *OriginalLength* field described in "MQMD - Message descriptor" on page 1705. The initial value of this field is MQOL_UNDEFINED.

StrucId (MQCHAR4):

The value must be:

MQMDE_STRUC_ID
Identifier for message descriptor extension structure.

For the C programming language, the constant MQMDE_STRUC_ID_ARRAY is also defined; this has the same value as MQMDE_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQMDE_STRUC_ID.

StrucLength (MQLONG):

This is the length of the MQMDE structure; the following value is defined:

MQMDE_LENGTH_2
Length of version-2 message descriptor extension structure.

The initial value of this field is MQMDE_LENGTH_2.

Version (MQLONG):

This is the structure version number; the value must be:

MQMDE_VERSION_2
Version-2 message descriptor extension structure.

The following constant specifies the version number of the current version:

MQMDE_CURRENT_VERSION
Current version of message descriptor extension structure.

The initial value of this field is MQMDE_VERSION_2.

Initial values and language declarations for MQMDE:

Table 166. Initial values of fields in MQMDE for MQMDE

Field name	Name of constant	Value of constant
StrucId	MQMDE_STRUC_ID	'MDE¬'
Version	MQMDE_VERSION_2	2
StrucLength	MQMDE_LENGTH_2	72
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQMDEF_NONE	0
GroupId	MQGI_NONE	Nulls
MsgSeqNumber	None	1
Offset	None	0
MsgFlags	MQMF_NONE	0
OriginalLength	MQOL_UNDEFINED	-1

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQMDE_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQMDE MyMDE = {MQMDE_DEFAULT};
```

C declaration:

```
typedef struct tagMQMDE MQMDE;
struct tagMQMDE {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    StrucLength;     /* Length of MQMDE structure */
  MQLONG    Encoding;        /* Numeric encoding of data that follows
                                MQMDE */
  MQLONG    CodedCharSetId;  /* Character-set identifier of data that
                                follows MQMDE */
  MQCHAR8   Format;          /* Format name of data that follows
                                MQMDE */
  MQLONG    Flags;           /* General flags */
  MQBYTE24  GroupId;         /* Group identifier */
  MQLONG    MsgSeqNumber;    /* Sequence number of logical message
                                within group */
```

```
  MQLONG    Offset;          /* Offset of data in physical message from
                                start of logical message */
  MQLONG    MsgFlags;        /* Message flags */
  MQLONG    OriginalLength;  /* Length of original message */
};
```

COBOL declaration:

```
**   MQMDE structure
  10 MQMDE.
**    Structure identifier
   15 MQMDE-STRUCID        PIC X(4).
**    Structure version number
   15 MQMDE-VERSION        PIC S9(9) BINARY.
**    Length of MQMDE structure
   15 MQMDE-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding of data that follows MQMDE
   15 MQMDE-ENCODING       PIC S9(9) BINARY.
**    Character-set identifier of data that follows MQMDE
   15 MQMDE-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows MQMDE
   15 MQMDE-FORMAT         PIC X(8).
**    General flags
   15 MQMDE-FLAGS          PIC S9(9) BINARY.
**    Group identifier
   15 MQMDE-GROUPID        PIC X(24).
**    Sequence number of logical message within group
   15 MQMDE-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Offset of data in physical message from start of logical message
   15 MQMDE-OFFSET         PIC S9(9) BINARY.
**    Message flags
   15 MQMDE-MSGFLAGS       PIC S9(9) BINARY.
**    Length of original message
   15 MQMDE-ORIGINALLENGTH PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQMDE based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Length of MQMDE structure */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows MQMDE */
  3 CodedCharSetId fixed bin(31), /* Character-set identifier of data
                                     that follows MQMDE */
  3 Format         char(8),       /* Format name of data that follows
                                     MQMDE */
  3 Flags          fixed bin(31), /* General flags */
  3 GroupId        char(24),      /* Group identifier */
  3 MsgSeqNumber   fixed bin(31), /* Sequence number of logical message
                                     within group */
  3 Offset         fixed bin(31), /* Offset of data in physical message
                                     from start of logical message */
  3 MsgFlags       fixed bin(31), /* Message flags */
  3 OriginalLength fixed bin(31); /* Length of original message */
```

High Level Assembler declaration:

```
MQMDE                 DSECT
MQMDE_STRUCID         DS   CL4   Structure identifier
MQMDE_VERSION         DS   F     Structure version number
MQMDE_STRUCLENGTH     DS   F     Length of MQMDE structure
MQMDE_ENCODING        DS   F     Numeric encoding of data that follows
*                                MQMDE
MQMDE_CODEDCHARSETID  DS   F     Character-set identifier of data that
*                                follows MQMDE
MQMDE_FORMAT          DS   CL8   Format name of data that follows MQMDE
MQMDE_FLAGS           DS   F     General flags
MQMDE_GROUPID         DS   XL24  Group identifier
MQMDE_MSGSEQNUMBER    DS   F     Sequence number of logical message
*                                within group
MQMDE_OFFSET          DS   F     Offset of data in physical message from
*                                start of logical message
MQMDE_MSGFLAGS        DS   F     Message flags
MQMDE_ORIGINALLENGTH  DS   F     Length of original message
*
MQMDE_LENGTH          EQU  *-MQMDE
                      ORG  MQMDE
MQMDE_AREA            DS   CL(MQMDE_LENGTH)
```

Visual Basic declaration:

```
Type MQMDE
  StrucId        As String*4 'Structure identifier'
  Version        As Long     'Structure version number'
  StrucLength    As Long     'Length of MQMDE structure'
  Encoding       As Long     'Numeric encoding of data that follows'
                             'MQMDE'
  CodedCharSetId As Long     'Character-set identifier of data that'
                             'follows MQMDE'
  Format         As String*8 'Format name of data that follows MQMDE'
  Flags          As Long     'General flags'
  GroupId        As MQBYTE24 'Group identifier'
  MsgSeqNumber   As Long     'Sequence number of logical message within'
                             'group'
  Offset         As Long     'Offset of data in physical message from'
                             'start of logical message'
  MsgFlags       As Long     'Message flags'
  OriginalLength As Long     'Length of original message'
End Type
```

MQMHBO - Message handle to buffer options:

The following table summarizes the fields in the structure. MQMHBO structure - message handle to buffer options

Table 167. Fields in MQMHBO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options controlling the action of MQMHBUF	Options

Overview for MQMHBO:

Availability: All WebSphere MQ systems and WebSphere MQ MQI clients.

Purpose: The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles. The structure is an input parameter on the MQMHBUF call.

Character set and encoding: Data in MQMHBO must be in the character set of the application and encoding of the application (MQENC_NATIVE).

Fields for MQMHBO:

Message handle to buffer options structure - fields

The MQMHBO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Message handle to buffer options structure - Options field

These options control the action of MQMHBUF.

You must specify the following option:

MQMHBO_PROPERTIES_IN_MQRFH2
When converting properties from a message handle into a buffer, convert them into the MQRFH2 format.

Optionally, you can also specify the following value. If required values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQMHBO_DELETE_PROPERTIES
Properties that are added to the buffer are deleted from the message handle. If the call fails no properties are deleted.

This is always an input field. The initial value of this field is MQMHBO_PROPERTIES_IN_MQRFH2.

StrucId (MQCHAR4):

Message handle to buffer options structure - StrucId field

This is the structure identifier. The value must be:

MQMHBO_STRUC_ID
Identifier for message handle to buffer options structure.

For the C programming language, the constant MQMHBO_STRUC_ID_ARRAY is also defined; this has the same value as MQMHBO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQMHBO_STRUC_ID.

Version (MQLONG):

Message handle to buffer options structure - Version field

This is the structure version number. The value must be:

MQMHBO_VERSION_1
Version number for message handle to buffer options structure.

The following constant specifies the version number of the current version:

MQMHBO_CURRENT_VERSION
Current version of message handle to buffer options structure.

This is always an input field. The initial value of this field is MQMHBO_VERSION_1.

Initial values and language declarations for MQMHBO:

Message handle to buffer structure - Initial values

Table 168. Initial values of fields in MQMHBO

Field name	Name of constant	Value of constant
StrucId	MQMHBO_STRUC_ID	'MHBO'
Version	MQMHBO_VERSION_1	1
Options	MQMHBO_PROPERTIES_IN_MQRFH2	

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQMHBO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQMHBO MyMHBO = {MQMHBO_DEFAULT};
```

C declaration:

Message handle to buffer options structure - C language declaration

```
typedef struct tagMQMHBO MQMHBO;
struct tagMQMHBO {
  MQCHAR4  StrucId;        /* Structure identifier */
  MQLONG   Version;        /* Structure version number */
  MQLONG   Options;        /* Options that control the action of
                              MQMHBUF */
};
```

COBOL declaration:

Message handle to buffer options structure - COBOL language declaration

```
**   MQMHBO structure
  10 MQMHBO.
**    Structure identifier
   15 MQMHBO-STRUCID               PIC X(4).
**    Structure version number
   15 MQMHBO-VERSION               PIC S9(9) BINARY.
**    Options that control the action of MQMHBUF
   15 MQMHBO-OPTIONS               PIC S9(9) BINARY.
```

PL/I declaration:

Message handle to buffer options structure - PL/I language declaration

```
Dcl
 1 MQMHBO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action
                                     of MQMHBUF */
```

High Level Assembler declaration:

Message handle to buffer options structure - Assembler language declaration

```
MQMHBO                  DSECT
MQMHBO_STRUCID          DS   CL4  Structure identifier
MQMHBO_VERSION          DS   F    Structure version number
MQMHBO_OPTIONS          DS   F    Options that control the
*                                  action of MQMHBUF
MQMHBO_LENGTH           EQU  *-MQMHBO
MQMHBO_AREA             DS   CL(MQMHBO_LENGTH)
```

MQOD - Object descriptor:

The following table summarizes the fields in the structure.

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
ObjectType	Object type	ObjectType
ObjectName	Object name	ObjectName
ObjectQMgrName	Object queue manager name	ObjectQMgrName
DynamicQName	Dynamic queue name	DynamicQName
AlternateUserId	Alternate user identifier	AlternateUserId
Note: The remaining fields are ignored if *Version* is less than MQOD_VERSION_2.		
RecsPresent	Number of object records present	RecsPresent
KnownDestCount	Number of local queues opened successfully	KnownDestCount
UnknownDestCount	Number of remote queues opened successfully	UnknownDestCount
InvalidDestCount	Number of queues that failed to open	InvalidDestCount
ObjectRecOffset	Offset of first object record from start of MQOD	ObjectRecOffset
ResponseRecOffset	Offset of first response record from start of MQOD	ResponseRecOffset
ObjectRecPtr	Address of first object record	ObjectRecPtr

Field	Description	Topic
ResponseRecPtr	Address of first response record	ResponseRecPtr
Note: The remaining fields are ignored if *Version* is less than MQOD_VERSION_3.		
AlternateSecurityId	Alternate security identifier	AlternateSecurityId
ResolvedQName	Resolved queue name	ResolvedQName
ResolvedQMgrName	Resolved queue manager name	ResolvedQMgrName
Note: The remaining fields are ignored if *Version* is less than MQOD_VERSION_4.		
ObjectString	Long object name	ObjectString
SelectionString	Selection string	SelectionString
ResObjectString	Resolved long object name	ResObjectString
ResolvedType	Resolved object type	ResolvedType

Overview for MQOD:

Availability: All WebSphere MQ systems, plus WebSphere MQ MQI clients connected to those systems.

Purpose: The MQOD structure is used to specify an object by name. The following types of object are valid:

- Queue or distribution list
- Namelist
- Process definition
- Queue manager
- Topic

The structure is an input/output parameter on the MQOPEN and MQPUT1 calls.

Version: The current version of MQOD is MQOD_VERSION_4. Applications that you want to port between several environments must ensure that the required version of MQOD is supported in all the environments concerned. Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions that follow.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQOD that is supported by the environment, but with the initial value of the *Version* field set to MQOD_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

To open a distribution list, *Version* must be MQOD_VERSION_2 or greater.

Character set and encoding: Data in MQOD must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQOD:

The MQOD structure contains the following fields; the fields are described in **alphabetical order**:

AlternateSecurityId (MQBYTE40):

This is a security identifier that is passed with the *AlternateUserId* to the authorization service to allow appropriate authorization checks to be performed. *AlternateSecurityId* is used only if:

- MQOO_ALTERNATE_USER_AUTHORITY is specified on the MQOPEN call, or
- MQPMO_ALTERNATE_USER_AUTHORITY is specified on the MQPUT1 call,

and the *AlternateUserId* field is not entirely blank up to the first null character or the end of the field.

On Windows, *AlternateSecurityId* can be used to supply the Windows security identifier (SID) that uniquely identifies the *AlternateUserId*. The SID for a user can be obtained from the Windows system by use of the `LookupAccountName()` Windows API call.

On z/OS, this field is ignored.

The *AlternateSecurityId* field has the following structure:

- The first byte is a binary integer containing the length of the significant data that follows; the value excludes the length byte itself. If no security identifier is present, the length is zero.
- The second byte indicates the type of security identifier that is present; the following values are possible:

  MQSIDT_NT_SECURITY_ID
  Windows security identifier.

  MQSIDT_NONE
  No security identifier.
- The third and subsequent bytes up to the length defined by the first byte contain the security identifier itself.
- Remaining bytes in the field are set to binary zero.

You can use the following special value:

MQSID_NONE
No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQSID_NONE_ARRAY is also defined; this has the same value as MQSID_NONE, but is an array of characters instead of a string.

This is an input field. The length of this field is given by MQ_SECURITY_ID_LENGTH. The initial value of this field is MQSID_NONE. This field is ignored if *Version* is less than MQOD_VERSION_3.

AlternateUserId (MQCHAR12):

If you specify MQOO_ALTERNATE_USER_AUTHORITY for the MQOPEN call, or MQPMO_ALTERNATE_USER_AUTHORITY for the MQPUT1 call, this field contains an alternative user identifier that is used to check the authorization for the open, in place of the user identifier that the application is currently running under. Some checks, however, are still carried out with the current user identifier (for example, context checks).

If MQOO_ALTERNATE_USER_AUTHORITY or MQPMO_ALTERNATE_USER_AUTHORITY is specified and this field is entirely blank up to the first null character or the end of the field, the open can succeed only if no user authorization is needed to open this object with the options specified.

If neither MQOO_ALTERNATE_USER_AUTHORITY nor MQPMO_ALTERNATE_USER_AUTHORITY is specified, this field is ignored.

The following differences exist in the environments indicated:

- On z/OS, only the first 8 characters of *AlternateUserId* are used to check the authorization for the open. However, the current user identifier must be authorized to specify this particular alternative user identifier; all 12 characters of the alternative user identifier are used for this check. The user identifier must contain only characters allowed by the external security manager.

  If *AlternateUserId* is specified for a queue, the value can be used subsequently by the queue manager when messages are put. If the MQPMO_*_CONTEXT options specified on the MQPUT or MQPUT1 call cause the queue manager to generate the identity context information, the queue manager places the *AlternateUserId* into the *UserIdentifier* field in the MQMD of the message, in place of the current user identifier.
- In other environments, *AlternateUserId* is used only for access control checks on the object being opened. If the object is a queue, *AlternateUserId* does not affect the content of the *UserIdentifier* field in the MQMD of messages sent using that queue handle.

This is an input field. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

DynamicQName (MQCHAR48):

This is the name of a dynamic queue that is to be created by the MQOPEN call. This is of relevance only when *ObjectName* specifies the name of a model queue; in all other cases *DynamicQName* is ignored.

The characters that are valid in the name are the same as those for *ObjectName*, except that an asterisk is also valid. A name that is blank (or one in which only blanks occur before the first null character) is not valid if *ObjectName* is the name of a model queue.

If the last nonblank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique at the local queue manager. To allow a sufficient number of characters for this, the asterisk is valid only in positions 1 through 33. There must be no characters other than blanks or a null character following the asterisk.

It is valid for the asterisk to occur in the first character position, in which case the name consists solely of the characters generated by the queue manager.

On z/OS, do not use a name with the asterisk in the first character position, as there can be no security checks made on a queue with a full name that is generated automatically.

This is an input field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is determined by the environment:

- On z/OS, the value is `'CSQ.*'`.
- On other platforms, the value is `'AMQ.*'`.

The value is a null-terminated string in C, and a blank-padded string in other programming languages.

InvalidDestCount (MQLONG):

This is the number of queues in the distribution list that failed to open successfully. If present, this field is also set when opening a single queue that is not in a distribution list.

Note: If present, this field is set *only* if the *CompCode* parameter on the MQOPEN or MQPUT1 call is MQCC_OK or MQCC_WARNING; it is *not* set if the *CompCode* parameter is MQCC_FAILED.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_1.

KnownDestCount (MQLONG):

This is the number of queues in the distribution list that resolve to local queues and that were opened successfully. The count does not include queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). If present, this field is also set when opening a single queue that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_1.

ObjectName (MQCHAR48):

This is the local name of the object as defined on the queue manager identified by *ObjectQMgrName*. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS:
  - Avoid names that begin or end with an underscore; they cannot be processed by the operations and control panels.
  - The percent character has a special meaning to RACF. If RACF is used as the external security manager, names must not contain the percent. If they do, those names are not included in any security checks when RACF generic profiles are used.
- On IBM i, names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see "Using topic strings" on page 1876.

The following points apply to the types of object indicated:

- If *ObjectName* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ObjectName* field the name of the queue created. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If *ObjectName* is the name of an alias queue with TARGTYPE(TOPIC), a security check is first made on the named alias queue; this is normal when alias queues are used. When the security check completes

successfully, the MQOPEN call will continue and will behave like an MQOPEN call on an MQOT_TOPIC; this includes making a security check against the administrative topic object.

- If *ObjectName* and *ObjectQMgrName* identify a shared queue owned by the queue-sharing group to which the local queue manager belongs, there must not also be a queue definition of the same name on the local queue manager. If there is such a definition (a local queue, alias queue, remote queue, or model queue), the call fails with reason code MQRC_OBJECT_NOT_UNIQUE.
- If the object being opened is a distribution list (that is, *RecsPresent* is present and greater than zero), *ObjectName* must be blank or the null string. If this condition is not satisfied, the call fails with reason code MQRC_OBJECT_NAME_ERROR.
- If *ObjectType* is MQOT_Q_MGR, special rules apply; in this case the name must be entirely blank up to the first null character or the end of the field.

This is an input/output field for the MQOPEN call when *ObjectName* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48):

This is the name of the queue manager on which the *ObjectName* object is defined. The characters that are valid in the name are the same as those for *ObjectName* (see "ObjectName (MQCHAR48)" on page 1772). A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

The following points apply to the types of object indicated:

- If *ObjectType* is MQOT_TOPIC, MQOT_NAMELIST, MQOT_PROCESS, or MQOT_Q_MGR, *ObjectQMgrName* must be blank or the name of the local queue manager.
- If *ObjectName* is the name of a model queue, the queue manager creates a dynamic queue with the attributes of the model queue, and returns in the *ObjectQMgrName* field the name of the queue manager on which the queue is created; this is the name of the local queue manager. A model queue can be specified only on the MQOPEN call; a model queue is not valid on the MQPUT1 call.
- If *ObjectName* is the name of a cluster queue, and *ObjectQMgrName* is blank, the destination of messages sent using the queue handle returned by the MQOPEN call is chosen by the queue manager (or cluster workload exit, if one is installed) as follows:
  - If MQOO_BIND_ON_OPEN is specified, the queue manager selects a particular instance of the cluster queue while processing the MQOPEN call, and all messages put using this queue handle are sent to that instance.
  - If MQOO_BIND_NOT_FIXED is specified, the queue manager can choose a different instance of the destination queue (residing on a different queue manager in the cluster) for each successive MQPUT call that uses this queue handle.

  If the application needs to send a message to a *specific* instance of a cluster queue (that is, a queue instance that resides on a particular queue manager in the cluster), the application must specify the name of that queue manager in the *ObjectQMgrName* field. This forces the local queue manager to send the message to the specified destination queue manager.
- If *ObjectName* is the name of a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ObjectQMgrName* can be the name of the queue-sharing group, the name of the local queue manager, or blank; the message is placed on the same queue whichever of these values is specified.

  Queue-sharing groups are supported only on z/OS.
- If *ObjectName* is the name of a shared queue that is owned by a remote queue-sharing group (that is, a queue-sharing group to which the local queue manager does *not* belong), *ObjectQMgrName* muat be the name of the queue-sharing group. You can use the name of a queue manager that belongs to that group, but this can delay the message if that particular queue manager is not available when the message arrives at the queue-sharing group.

- If the object being opened is a distribution list (that is, *RecsPresent* is greater than zero), *ObjectQMgrName* must be blank or the null string. If this condition is not satisfied, the call fails with reason code MQRC_OBJECT_Q_MGR_NAME_ERROR.

This is an input/output field for the MQOPEN call when *ObjectName* is the name of a model queue, and an input-only field in all other cases. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectRecOffset (MQLONG):

This is the offset in bytes of the first MQOR object record from the start of the MQOD structure. The offset can be positive or negative. *ObjectRecOffset* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

When a distribution list is being opened, an array of one or more MQOR object records must be provided in order to specify the names of the destination queues in the distribution list. This can be done in one of two ways:

- By using the offset field *ObjectRecOffset*.

  In this case, the application must declare its own structure containing an MQOD followed by the array of MQOR records (with as many array elements as are needed), and set *ObjectRecOffset* to the offset of the first element in the array from the start of the MQOD. Ensure that this offset is correct and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

  Use *ObjectRecOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a way that is not portable to different environments (for example, the COBOL programming language).
- By using the pointer field *ObjectRecPtr*.

  In this case, the application can declare the array of MQOR structures separately from the MQOD structure, and set *ObjectRecPtr* to the address of the array.

  Use *ObjectRecPtr* for programming languages that support the pointer data type in a way that is portable to different environments (for example, the C programming language).

Whatever technique you choose, use one of *ObjectRecOffset* and *ObjectRecPtr*; the call fails with reason code MQRC_OBJECT_RECORDS_ERROR if both are zero, or both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ObjectRecPtr (MQPTR):

This is the address of the first MQOR object record. *ObjectRecPtr* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

You can use either *ObjectRecPtr* or *ObjectRecOffset* to specify the object records, but not both; see the description of the *ObjectRecOffset* field above for details. If you do not use *ObjectRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

ObjectString (MQCHARV):

The ObjectString field specifies the long object name.

This specifies the long object name to be used. This field is only referenced for certain values of *ObjectType*, and is ignored for all other values. See the description of *ObjectType* for details of which values indicate that this field is used.

If *ObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_OBJECT_STRING_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see "Using topic strings" on page 1876.

ObjectType (MQLONG):

The type of object being named in the object descriptor. Possible values are:

MQOT_CLNTCONN_CHANNEL
Client connection channel. The name of the object is found in the *ObjectName* field.

MQOT_Q
Queue. The name of the object is found in the *ObjectName* field.

MQOT_NAMELIST
Namelist. The name of the object is found in the *ObjectName* field

MQOT_PROCESS
Process definition. The name of the object is found in the *ObjectName* field

MQOT_Q_MGR
Queue manager. The name of the object is found in the *ObjectName* field

MQOT_TOPIC
Topic. The full topic name can be built from two different fields: *ObjectName* and *ObjectString*.

For details of how those two fields are used, see "Using topic strings" on page 1876.

This is always an input field. The initial value of this field is MQOT_Q.

RecsPresent (MQLONG):

This is the number of MQOR object records that have been provided by the application. If this number is greater than zero, it indicates that a distribution list is being opened, with *RecsPresent* being the number of destination queues in the list. A distribution list can contain only one destination.

The value of *RecsPresent* must not be less than zero, and if it is greater than zero *ObjectType* must be MQOT_Q; the call fails with reason code MQRC_RECS_PRESENT_ERROR if these conditions are not satisfied.

On z/OS, this field must be zero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ResObjectString (MQCHARV):

The ResObjectString field is the long object name after the queue manager resolves the name provided in the *ObjectName* field.

This field is returned only for topics and queue aliases that reference a topic object.

If the long object name is provided in *ObjectString* and nothing is provided in *ObjectName*, then the value returned in this field is the same as provided in *ObjectString*.

If this field is omitted (that is ResObjectString.VSBufSize is zero) then the *ResObjectString* will not be returned, but the length will be returned in ResObjectString.VSLength.

If the buffer length (provided in ResObjectStrng.VSBufSize) is shorter than the full *ResObjectString*, the string will be truncated and will return as many of the rightmost characters as can fit in the provided buffer.

If *ResObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_RES_OBJECT_STRING_ERROR.

ResolvedQMgrName (MQCHAR48):

This is the name of the destination queue manager after the local queue manager resolves the name. The name returned is the name of the queue manager that owns the queue identified by *ResolvedQName*. *ResolvedQMgrName* can be the name of the local queue manager.

If *ResolvedQName* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ResolvedQMgrName* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ResolvedQName* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ResolvedQMgrName* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output
- A cluster queue with MQOO_BIND_NOT_FIXED specified (or with MQOO_BIND_AS_Q_DEF in effect when the *DefBind* queue attribute has the value MQBND_BIND_NOT_FIXED)
- A distribution list

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *Version* is less than MQOD_VERSION_3.

ResolvedQName (MQCHAR48):

This is the name of the destination queue after the local queue manager resolves the name. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

A nonblank value is returned only if the object is a single queue opened for browse, input, or output (or any combination). If the object opened is any of the following, *ResolvedQName* is set to blanks:

- Not a queue
- A queue, but not opened for browse, input, or output

- A distribution list
- An alias queue that references a topic object (refer to ResObjectString instead).
- An alias queue that resolves to a topic object.

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages. This field is ignored if *Version* is less than MQOD_VERSION_3.

ResolvedType (MQLONG):

The type of the resolved (base) object being opened.

The possible values are:

MQOT_Q
The resolved object is a queue. This value applies when a queue is opened directly or when an alias queue pointing to a queue is opened.

MQOT_TOPIC
The resolved object is a topic. This value applies when a topic is opened directly or when an alias queue pointing to a topic object is opened.

MQOT_NONE
The resolved type is neither a queue nor a topic.

ResponseRecOffset (MQLONG):

This is the offset in bytes of the first MQRR response record from the start of the MQOD structure. The offset can be positive or negative. *ResponseRecOffset* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

When a distribution list is being opened, you can provide an array of one or more MQRR response records in order to identify the queues that failed to open (*CompCode* field in MQRR), and the reason for each failure (*Reason* field in MQRR). The data is returned in the array of response records in the same order as the queue names occur in the array of object records. The queue manager sets the response records only when the outcome of the call is mixed (that is, some queues were opened successfully while others failed, or all failed but for different reasons); reason code MQRC_MULTIPLE_REASONS from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the *Reason* parameter of the MQOPEN or MQPUT1 call, and the response records are not set. Response records are optional, but if they are supplied there must be *RecsPresent* of them.

The response records can be provided in the same way as the object records, either by specifying an offset in *ResponseRecOffset*, or by specifying an address in *ResponseRecPtr*; see the description of *ObjectRecOffset* above for details of how to do this. However, no more than one of *ResponseRecOffset* and *ResponseRecPtr* can be used; the call fails with reason code MQRC_RESPONSE_RECORDS_ERROR if both are nonzero.

For the MQPUT1 call, these response records are used to return information about errors that occur when the message is sent to the queues in the distribution list, as well as errors that occur when the queues are opened. The completion code and reason code from the put operation for a queue replace those from the open operation for that queue only if the completion code from the latter was MQCC_OK or MQCC_WARNING.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_2.

ResponseRecPtr (MQPTR):

This is the address of the first MQRR response record. *ResponseRecPtr* is used only when a distribution list is being opened. The field is ignored if *RecsPresent* is zero.

Use either *ResponseRecPtr* or *ResponseRecOffset* to specify the response records, but not both; see the description of the *ResponseRecOffset* field above for details. If you do not use *ResponseRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQOD_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

SelectionString (MQCHARV):

This is the string used to provide the selection criteria used when retrieving messages off a queue.

SelectionString must not be provided in the following cases:

- If *ObjectType* is not MQOT_Q
- If the queue being opened is not being opened using one of the MQOO_BROWSE, or MQOO_INPUT_* options

If *SelectionString* is provided in these cases, the call fails with reason code MQRC_SELECTOR_INVALID_FOR_TYPE.

If *SelectionString* is specified incorrectly, according to the description of how to use the "MQCHARV - Variable Length String" on page 1581 structure, or if it exceeds the maximum length, the call fails with reason code MQRC_SELECTION_STRING_ERROR. The maximum length of *SelectionString* is MQ_SELECTOR_LENGTH.

SelectionString usage is described in Selectors.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQOD_STRUC_ID
Identifier for object descriptor structure.

For the C programming language, the constant MQOD_STRUC_ID_ARRAY is also defined; this has the same value as MQOD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQOD_STRUC_ID.

UnknownDestCount (MQLONG):

This is the number of queues in the distribution list that resolve to remote queues and that were opened successfully. If present, this field is also set when opening a single queue that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is ignored if *Version* is less than MQOD_VERSION_1.

Version (MQLONG):

This is the structure version number; the value must be one of the following:

MQOD_VERSION_1
Version-1 object descriptor structure.

MQOD_VERSION_2
Version-2 object descriptor structure.

MQOD_VERSION_3
Version-3 object descriptor structure.

MQOD_VERSION_4
Version-4 object descriptor structure.

All versions are supported in all WebSphere MQ V7.0 environments.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQOD_CURRENT_VERSION
Current version of object descriptor structure.

This is always an input field. The initial value of this field is MQOD_VERSION_1.

Initial values and language declarations for MQOD:

Field name	Name of constant	Value of constant
StrucId	MQOD_STRUC_ID	'OD¬¬'
Version	MQOD_VERSION_1	1
ObjectType	MQOT_Q	1
ObjectName	None	Null string or blanks
ObjectQMgrName	None	Null string or blanks
DynamicQName	None	'CSQ.*' on z/OS; 'AMQ.*' otherwise
AlternateUserId	None	Null string or blanks
RecsPresent	None	0
KnownDestCount	None	0
UnknownDestCount	None	0
InvalidDestCount	None	0
ObjectRecOffset	None	0
ResponseRecOffset	None	0
ObjectRecPtr	None	Null pointer or null bytes
ResponseRecPtr	None	Null pointer or null bytes
AlternateSecurityId	MQSID_NONE	Nulls
ResolvedQName	None	Null string or blanks
ResolvedQMgrName	None	Null string or blanks
ObjectString	MQCHARV_DEFAULT	As defined for MQCHARV
SelectionString	MQCHARV_DEFAULT	As defined for MQCHARV
ResObjectString	MQCHARV_DEFAULT	As defined for MQCHARV

Field name	Name of constant	Value of constant
ResolvedType	MQOT_NONE	0

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQOD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

   ```
   MQOD MyOD = {MQOD_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQOD MQOD;
struct tagMQOD {
  MQCHAR4    StrucId;              /* Structure identifier */
  MQLONG     Version;              /* Structure version number */
  MQLONG     ObjectType;           /* Object type */
  MQCHAR48   ObjectName;           /* Object name */
  MQCHAR48   ObjectQMgrName;       /* Object queue manager name */
  MQCHAR48   DynamicQName;         /* Dynamic queue name */
  MQCHAR12   AlternateUserId;      /* Alternate user identifier */
  /* Ver:1 */
  MQLONG     RecsPresent;          /* Number of object records present */
  MQLONG     KnownDestCount;       /* Number of local queues opened
                                      successfully */
  MQLONG     UnknownDestCount;     /* Number of remote queues opened
                                      successfully */
  MQLONG     InvalidDestCount;     /* Number of queues that failed to
                                      open */
  MQLONG     ObjectRecOffset;      /* Offset of first object record from
                                      start of MQOD */
  MQLONG     ResponseRecOffset;    /* Offset of first response record
                                      from start of MQOD */
  MQPTR      ObjectRecPtr;         /* Address of first object record */
  MQPTR      ResponseRecPtr;       /* Address of first response record */
  /* Ver:2 */
  MQBYTE40   AlternateSecurityId;  /* Alternate security identifier */
  MQCHAR48   ResolvedQName;        /* Resolved queue name */
  MQCHAR48   ResolvedQMgrName;     /* Resolved queue manager name */
  /* Ver:3 */
  MQCHARV    ObjectString;         /* Object Long name */
  MQCHARV    SelectionString;      /* Message Selector */
  MQCHARV    ResObjectString;      /* Resolved Long object name*/
  MQLONG     ResolvedType          /* Alias queue resolved
                                      oject type */
  /* Ver:4 */
};
```

COBOL declaration:

```
** MQOD structure
 10  MQOD.
** Structure identifier
 15  MQOD-STRUCID                    PIC X(4).
** Structure version number
 15  MQOD-VERSION                    PIC S9(9) BINARY.
** Object type
 15  MQOD-OBJECTTYPE                 PIC S9(9) BINARY.
** Object name
 15  MQOD-OBJECTNAME                 PIC X(48).
** Object queue manager name
 15  MQOD-OBJECTQMGRNAME             PIC X(48).
** Dynamic queue name
 15  MQOD-DYNAMICQNAME               PIC X(48).
** Alternate user identifier
 15  MQOD-ALTERNATEUSERID            PIC X(12).
** Number of object records present
 15  MQOD-RECSPRESENT                PIC S9(9) BINARY.
** Number of local queues opened successfully
 15  MQOD-KNOWNDESTCOUNT             PIC S9(9) BINARY.
** Number of remote queues opened successfully
 15  MQOD-UNKNOWNDESTCOUNT           PIC S9(9) BINARY.
** Number of queues that failed to open
 15  MQOD-INVALIDDESTCOUNT           PIC S9(9) BINARY.
** Offset of first object record from start of MQOD
 15  MQOD-OBJECTRECOFFSET            PIC S9(9) BINARY.
** Offset of first response record from start of MQOD
 15  MQOD-RESPONSERECOFFSET          PIC S9(9) BINARY.
** Address of first object record
 15  MQOD-OBJECTRECPTR               POINTER.
** Address of first response record
 15  MQOD-RESPONSERECPTR             POINTER.
** Alternate security identifier
 15  MQOD-ALTERNATESECURITYID        PIC X(40).
** Resolved queue name
 15  MQOD-RESOLVEDQNAME              PIC X(48).
** Resolved queue manager name
 15  MQOD-RESOLVEDQMGRNAME           PIC X(48).
** Object Long name
 15 MQOD-OBJECTSTRING.
** Address of variable length string
    20  MQOD-OBJECTSTRING-VSPTR       POINTER.
** Offset of variable length string
    20  MQOD-OBJECTSTRING-VSOFFSET    PIC S9(9) BINARY.
** size of buffer
    20  MQOD-OBJECTSTRING-VSBUFSIZE   PIC S9(9) BINARY.
** Length of variable length string
    20  MQOD-OBJECTSTRING-VSLENGTH    PIC S9(9) BINARY.
** CCSID of variable length string
    20  MQOD-OBJECTSTRING-VSCCSID     PIC S9(9) BINARY.
** Message Selector
 15 MQOD-SELECTIONSTRING.
** Address of variable length string
    20  MQOD-SELECTIONSTRING-VSPTR    POINTER.
** Offset of variable length string
    20  MQOD-SELECTIONSTRING-VSOFFSET PIC S9(9) BINARY.
** size of buffer
    20  MQOD-SELECTIONSTRING-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
    20  MQOD-SELECTIONSTRING-VSLENGTH PIC S9(9) BINARY.
** CCSID of variable length string
    20  MQOD-SELECTIONSTRING-VSCCSID  PIC S9(9) BINARY.
** Resolved Long object name
 15 MQOD-RESOBJECTSTRING.
** Address of variable length string
    20  MQOD-RESOBJECTSTRING-VSPTR    POINTER.
```

```
** Offset of variable length string
   20  MQOD-RESOBJECTSTRING-VSOFFSET  PIC S9(9) BINARY.
** size of buffer
   20  MQOD-RESOBJECTSTRING-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
   20  MQOD-RESOBJECTSTRING-VSLENGTH  PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQOD-RESOBJECTSTRING-VSCCSID   PIC S9(9) BINARY.
** Alias queue resolved object type
   15  MQOD-RESOLVEDTYPE              PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQOD based,
  3 StrucId             char(4),       /* Structure identifier */
  3 Version             fixed bin(31), /* Structure version number */
  3 ObjectType          fixed bin(31), /* Object type */
  3 ObjectName          char(48),      /* Object name */
  3 ObjectQMgrName      char(48),      /* Object queue manager name */
  3 DynamicQName        char(48),      /* Dynamic queue name */
  3 AlternateUserId     char(12),      /* Alternate user identifier */
  3 RecsPresent         fixed bin(31), /* Number of object records
                                          present */
  3 KnownDestCount      fixed bin(31), /* Number of local queues opened
                                          successfully */
  3 UnknownDestCount    fixed bin(31), /* Number of remote queues opened
                                          successfully */
  3 InvalidDestCount    fixed bin(31), /* Number of queues that failed to
                                          open */
  3 ObjectRecOffset     fixed bin(31), /* Offset of first object record
                                          from start of MQOD */
  3 ResponseRecOffset   fixed bin(31), /* Offset of first response record
                                          from start of MQOD */
  3 ObjectRecPtr        pointer,       /* Address of first object record */
  3 ResponseRecPtr      pointer,       /* Address of first response
                                          record */
  3 AlternateSecurityId char(40),      /* Alternate security identifier */
  3 ResolvedQName       char(48),      /* Resolved queue name */
  3 ResolvedQMgrName    char(48),      /* Resolved queue manager name */
  3 ObjectString,                      /* Object Long name */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31), /* CCSID of variable length string */
  3 SelectionString,                   /* Message Selection */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31), /* CCSID of variable length string */
  3 ResObjectString,                   /* Resolved Long object name */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31), /* CCSID of variable length string */
 3 ResolvedType        fixed bin(31); /* Alias queue resolved object type */
```

High Level Assembler declaration:

```
MQOD                           DSECT
MQOD_STRUCID                   DS   CL4   Structure identifier
MQOD_VERSION                   DS   F     Structure version number
MQOD_OBJECTTYPE                DS   F     Object type
MQOD_OBJECTNAME                DS   CL48  Object name
MQOD_OBJECTQMGRNAME            DS   CL48  Object queue manager name
MQOD_DYNAMICQNAME              DS   CL48  Dynamic queue name
MQOD_ALTERNATEUSERID           DS   CL12  Alternate user identifier
MQOD_RECSPRESENT               DS   F     Number of object records present
MQOD_KNOWNDESTCOUNT            DS   F     Number of local queues opened
*                                         successfully
MQOD_UNKNOWNDESTCOUNT          DS   F     Number of remote queues opened
*                                         successfully
MQOD_INVALIDDESTCOUNT          DS   F     Number of queues that failed to
*                                         open
MQOD_OBJECTRECOFFSET           DS   F     Offset of first object record from
*                                         start of MQOD
MQOD_RESPONSERECOFFSET         DS   F     Offset of first response record
*                                         from start of MQOD
MQOD_OBJECTRECPTR              DS   F     Address of first object record
MQOD_RESPONSERECPTR            DS   F     Address of first response record
MQOD_ALTERNATESECURITYID       DS   XL40  Alternate security identifier
MQOD_RESOLVEDQNAME             DS   CL48  Resolved queue name
MQOD_RESOLVEDQMGRNAME          DS   CL48  Resolved queue manager name
MQOD_OBJECTSTRING              DS   F     Object Long name
MQOD_OBJECTSTRING_VSPTR        DS   F     Address of variable length string
MQOD_OBJECTSTRING_VSOFFSET     DS   F     Offset of variable length string
MQOD_OBJECTSTRING_VSBUFSIZE    DS   F     size of buffer
MQOD_OBJECTSTRING_VSLENGTH     DS   F     Length of variable length string
MQOD_OBJECTSTRING_VSCCSID      DS   F     CCSID of variable length string
MQOD_OBJECTSTRING_LENGTH       EQU  *- MQOD_OBJECTSTRING
                               ORG  MQOD_OBJECTSTRING
MQOD_OBJECTSTRING_AREA         DS   CL(MQOD_OBJECTSTRING_LENGTH)
*
MQOD_SELECTIONSTRING           DS   F     Message Selector
MQOD_SELECTIONSTRING_VSPTR     DS   F     Address of variable length string
MQOD_SELECTIONSTRING_VSOFFSET  DS   F     Offset of variable length string
MQOD_SELECTIONSTRING_VSBUFSIZE DS   F     size of buffer
MQOD_SELECTIONSTRING_VSLENGTH  DS   F     Length of variable length string
MQOD_SELECTIONSTRING_VSCCSID   DS   F     CCSID of variable length string
MQOD_SELECTIONSTRING_LENGTH    EQU  *- MQOD_SELECTIONSTRING
                               ORG  MQOD_SELECTIONSTRING
MQOD_SELECTIONSTRING_AREA      DS   CL(MQOD_SELECTIONSTRING_LENGTH)
*
MQOD_RESOBJECTSTRING           DS   F     Resolved Long object name
MQOD_RESOBJECTSTRING_VSPTR     DS   F     Address of variable length string
MQOD_RESOBJECTSTRING_VSOFFSET  DS   F     Offset of variable length string
MQOD_RESOBJECTSTRING_VSBUFSIZE DS   F     size of buffer
MQOD_RESOBJECTSTRING_VSLENGTH  DS   F     Length of variable length string
MQOD_RESOBJECTSTRING_VSCCSID   DS   F     CCSID of variable length string
MQOD_RESOBJECTSTRING_LENGTH    EQU  *- MQOD_RESOBJECTSTRING
                               ORG  MQOD_RESOBJECTSTRING
MQOD_RESOBJECTSTRING_AREA      DS   CL(MQOD_RESOBJECTSTRING_LENGTH)
MQOD_RESOLVEDTYPE              DS   F     Alias queue object resolved type
*
MQOD_LENGTH                    EQU  *-MQOD
                               ORG  MQOD
MQOD_AREA                      DS   CL(MQOD_LENGTH)
```

Visual Basic declaration:

```
Type MQOD
  StrucId             As String*4  'Structure identifier'
  Version             As Long      'Structure version number'
  ObjectType          As Long      'Object type'
  ObjectName          As String*48 'Object name'
  ObjectQMgrName      As String*48 'Object queue manager name'
  DynamicQName        As String*48 'Dynamic queue name'
  AlternateUserId     As String*12 'Alternate user identifier'
  RecsPresent         As Long      'Number of object records present'
  KnownDestCount      As Long      'Number of local queues opened'
                                   'successfully'
  UnknownDestCount    As Long      'Number of remote queues opened'
                                   'successfully'
  InvalidDestCount    As Long      'Number of queues that failed to'
                                   'open'
  ObjectRecOffset     As Long      'Offset of first object record from'
                                   'start of MQOD'
  ResponseRecOffset   As Long      'Offset of first response record'
                                   'from start of MQOD'
  ObjectRecPtr        As MQPTR     'Address of first object record'
  ResponseRecPtr      As MQPTR     'Address of first response record'
  AlternateSecurityId As MQBYTE40  'Alternate security identifier'
  ResolvedQName       As String*48 'Resolved queue name'
  ResolvedQMgrName    As String*48 'Resolved queue manager name'
End Type
```

MQOR - Object record:

The following table summarizes the fields in the structure.

Table 169. Fields in MQOR

Field	Description	Topic
ObjectName	Object name	ObjectName
ObjectQMgrName	Object queue manager name	ObjectQMgrName

Overview for MQOR:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

Purpose: Use the MQOR structure to specify the queue name and queue-manager name of a single destination queue. MQOR is an input structure for the MQOPEN and MQPUT1 calls.

Character set and encoding: Data in MQOR must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQOPEN call, you can open a list of queues; this list is called a *distribution list*. Each message put using the queue handle returned by that MQOPEN call is placed on each of the queues in the list, provided that the queue was opened successfully.

Fields for MQOR:

The MQOR structure contains the following fields; the fields are described in **alphabetical order**:

ObjectName (MQCHAR48):

This is the same as the *ObjectName* field in the MQOD structure (see MQOD for details), except that:

- It must be the name of a queue.
- It must not be the name of a model queue.

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48):

This is the same as the *ObjectQMgrName* field in the MQOD structure (see MQOD for details).

This is always an input field. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

Initial values and language declarations for MQOR:

Table 170. Initial values of fields in MQOR for MQOR

Field name	Name of constant	Value of constant
ObjectName	None	Null string or blanks
ObjectQMgrName	None	Null string or blanks

Notes:

1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
2. In the C programming language, the macro variable MQOR_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQOR MyOR = {MQOR_DEFAULT};
```

C declaration:

```
typedef struct tagMQOR MQOR;
struct tagMQOR {
  MQCHAR48  ObjectName;      /* Object name */
  MQCHAR48  ObjectQMgrName;  /* Object queue manager name */
};
```

COBOL declaration:

```
**   MQOR structure
  10 MQOR.
**    Object name
   15 MQOR-OBJECTNAME     PIC X(48).
**    Object queue manager name
   15 MQOR-OBJECTQMGRNAME PIC X(48).
```

PL/I declaration:

```
dcl
 1 MQOR based,
  3 ObjectName     char(48), /* Object name */
  3 ObjectQMgrName char(48); /* Object queue manager name */
```

Visual Basic declaration:

```
Type MQOR
  ObjectName     As String*48 'Object name'
  ObjectQMgrName As String*48 'Object queue manager name'
End Type
```

MQPD - Property descriptor:

The following table summarizes the fields in the structure.

Table 171. Fields in MQPD

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options
Support	Required support for message property	Support
Context	Message context to which property belongs	Context
CopyOptions	Copy options to which property belongs	CopyOptions

Overview for MQPD:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS and WebSphere MQ MQI clients.

Purpose: The **MQPD** is used to define the attributes of a property. The structure is an input/output parameter on the MQSETMP call and an output parameter on the MQINQMP call.

Character set and encoding: Data in **MQPD** must be in the character set of the application and encoding of the application (**MQENC_NATIVE**).

Fields for MQPD:

The MQPD structure contains the following fields; the fields are described in **alphabetical order**:

Context (MQLONG):

This describes what message context the property belongs to.

When a queue manager receives a message containing a WebSphere MQ-defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the *Context* field.

The following option can be specified:

MQPD_USER_CONTEXT
The property is associated with the user context.

No special authorization is required to be able to set a property associated with the user context using the MQSETMP call.

On a WebSphere MQ Version 7.0 queue manager, a property associated with the user context is saved as described for MQOO_SAVE_ALL_CONTEXT. An MQPUT call with MQPMO_PASS_ALL_CONTEXT specified, causes the property to be copied from the saved context into the new message.

If the option previously described is not required, the following option can be used:

MQPD_NO_CONTEXT
The property is not associated with a message context.

An unrecognized value is rejected with a *Reason*code of MQRC_PD_ERROR

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is MQPD_NO_CONTEXT.

CopyOptions (MQLONG):

This describes which type of messages the property should be copied into. This is an output only field for recognized WebSphere MQ defined properties; WebSphere MQ sets the appropriate value.

When a queue manager receives a message containing a WebSphere MQ defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the *CopyOptions* field.

You can specify one or more of these options, and if you need more than one, the values can be:

- Added (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQCOPY_FORWARD
This property is copied into a message being forwarded.

MQCOPY_PUBLISH
This property is copied into the message received by a subscriber when a message is being published.

MQCOPY_REPLY
This property is copied into a reply message.

MQCOPY_REPORT
This property is copied into a report message.

MQCOPY_ALL
This property is copied into all types of subsequent messages.

Default option: The following option can be specified to supply the default set of copy options:

MQCOPY_DEFAULT
This property is copied into a message being forwarded, into a report message, or into a message received by a subscriber when a message is being published.

This is equivalent to specifying the combination of options MQCOPY_FORWARD, plus MQCOPY_REPORT, plus MQCOPY_PUBLISH.

If none of the options described above is required, use the following option:

MQCOPY_NONE
Use this value to indicate that no other copy options are specified; programmatically no relationship exists between this property and subsequent messages. This is always returned for message descriptor properties.

This is an input/output field to the MQSETMP call and an output field from the MQINQMP call. The initial value of this field is MQCOPY_DEFAULT.

Options (MQLONG):

The value must be:

MQPD_NONE
No options specified

This is always an input field. The initial value of this field is MQPD_NONE.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQPD_STRUC_ID
Identifier for property descriptor structure.

For the C programming language, the constant **MQPD_STRUC_ID_ARRAY** is also defined; this has the same value as **MQPD_STRUC_ID**, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is **MQPD_STRUC_ID**.

Support (MQLONG):
This field describes what level of support for the message property is required of the queue manager, in order for the message containing this property to be put to a queue. This applies only to WebSphere MQ-defined properties; support for all other properties is optional.

The field is automatically set to the correct value when the WebSphere MQ-defined property is known by the queue manager. If the property is not recognized, MQPD_SUPPORT_OPTIONAL is assigned. When a queue manager receives a message containing a WebSphere MQ-defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the *Support* field.

When setting a WebSphere MQ-defined property using the MQSETMP call on a message handle where the MQCMHO_NO_VALIDATION option was set, *Support* becomes an input field. This allows an application to put a WebSphere MQ-defined property, with the correct value, where the property is unsupported by the connected queue manager, but where the message is intended to be processed on another queue manager.

The value MQPD_SUPPORT_OPTIONAL is always assigned to properties that are not WebSphere MQ-defined properties.

If a WebSphere MQ Version 7.0 queue manager, that supports message properties, receives a property that contains an unrecognized *Support* value, the property is treated as if:

- MQPD_SUPPORT_REQUIRED was specified if any of the unrecognized values are contained in the MQPD_REJECT_UNSUP_MASK.
- MQPD_SUPPORT_REQUIRED_IF_LOCAL was specified if any of the unrecognized values are contained in the MQPD_ACCEPT_UNSUP_IF_XMIT_MASK
- MQPD_SUPPORT_OPTIONAL was specified otherwise.

One of the following values is returned by the MQINQMP call, or one of the values can be specified, when using the MQSETMP call on a message handle where the MQCMHO_NO_VALIDATION option is set:

MQPD_SUPPORT_OPTIONAL
The property is accepted by a queue manager even if it is not supported. The property can be

discarded in order for the message to flow to a queue manager that does not support message properties. This value is also assigned to properties that are not WebSphere MQ-defined.

MQPD_SUPPORT_REQUIRED
Support for the property is required. The message is rejected by a queue manager that does not support the WebSphere MQ-defined property. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

MQPD_SUPPORT_REQUIRED_IF_LOCAL
The message is rejected by a queue manager that does not support the WebSphere MQ-defined property if the message is destined for a local queue. The MQPUT or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_UNSUPPORTED_PROPERTY.

The MQPUT or MQPUT1 call succeeds if the message is destined for a remote queue manager.

This is an output field on the MQINQMP call and an input field on the MQSETMP call if the message handle was created with the MQCMHO_NO_VALIDATION option set. The initial value of this field is MQPD_SUPPORT_OPTIONAL.

Version (MQLONG):

This is the structure version number; the value must be:

MQPD_VERSION_1
Version-1 property descriptor structure.

The following constant specifies the version number of the current version:

MQPD_CURRENT_VERSION
Current version of property descriptor structure.

This is always an input field. The initial value of this field is **MQPD_VERSION_1**.

Initial values and language declarations for MQPD:

Table 172. Initial values of fields in MQPD

Field name	Name of constant	Value of constant
StrucId	MQPD_STRUC_ID	'PD'
Version	MQPD_VERSION_1	1
Options	MQPD_NONE	0
Support	MQPD_SUPPORT_OPTIONAL	0
Context	MQPD_NO_CONTEXT	0
CopyOptions	MQCOPY_DEFAULT	0

Notes:

1. In the C programming language, the macro variable MQPD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQPD MyPD = {MQPD_DEFAULT};
```

C declaration:

```
typedef struct tagMQPD MQPD;
struct tagMQPD {
  MQCHAR4  StrucId;     /* Structure identifier */
  MQLONG   Version;     /* Structure version number */
  MQLONG   Options;     /* Options that control the action of
                           MQSETMP and MQINQMP */
  MQLONG   Support;     /* Property support option */
  MQLONG   Context;     /* Property context */
  MQLONG   CopyOptions; /* Property copy options */
};
```

COBOL declaration:

```
**   MQPD structure
  10 MQPD.
**    Structure identifier
   15 MQPD-STRUCID PIC X(4).
**    Structure version number
   15 MQPD-VERSION PIC S9(9) BINARY.
**    Options that control the action of MQSETMP and
**    MQINQMP
   15 MQPD-OPTIONS PIC S9(9) BINARY.
**    Property support option
   15 MQPD-SUPPORT PIC S9(9) BINARY.
**    Property context
   15 MQPD-CONTEXT PIC S9(9) BINARY.
**    Property copy options
   15 MQPD-COPYOPTIONS PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQPD based,
  3 StrucId     char(4),       /* Structure identifier */
  3 Version     fixed bin(31), /* Structure version number */
  3 Options     fixed bin(31), /* Options that control the action
                                  of MQSETMP and MQINQMP */
  3 Support     fixed bin(31), /* Property support option */
  3 Context     fixed bin(31), /* Property context */
  3 CopyOptions fixed bin(31); /* Property copy options */
```

High Level Assembler declaration:

```
MQPD             DSECT
MQPD_STRUCID     DS   CL4    Structure identifier
MQPD_VERSION     DS   F      Structure version number
MQPD_OPTIONS     DS   F      Options that control the
*                            action of MQSETMP and MQINQMP
MQPD_SUPPORT     DS   F      Property support option
MQPD_CONTEXT     DS   F      Property context
MQPD_COPYOPTIONS DS   F      Property copy options
MQPD_LENGTH      EQU  *-MQPD
MQPD_AREA        DS   CL(MQPD_LENGTH)
```

MQPMO - Put-message options:

The following table summarizes the fields in the structure.

Table 173. MQPMO structure

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options that control the action of MQPUT and MQPUT1	Options
Timeout	Reserved	Timeout
Context	Object handle of input queue	Context
KnownDestCount	Number of messages sent successfully to local queues	KnownDestCount
UnknownDestCount	Number of messages sent successfully to remote queues	UnknownDestCount
InvalidDestCount	Number of messages that could not be sent	InvalidDestCount
ResolvedQName	Resolved name of destination queue	ResolvedQName
ResolvedQMgrName	Resolved name of destination queue manager	ResolvedQMgrName
Note: The remaining fields are ignored if *Version* is less than MQPMO_VERSION_2.		
RecsPresent	Number of put message records or response records present	RecsPresent
PutMsgRecFields	Flags indicating which MQPMR fields are present	PutMsgRecFields
PutMsgRecOffset	Offset of first put-message record from start of MQPMO	PutMsgRecOffset
ResponseRecOffset	Offset of first response record from start of MQPMO	ResponseRecOffset
PutMsgRecPtr	Address of first put message record	PutMsgRecPtr
ResponseRecPtr	Address of first response record	ResponseRecPtr
Note: The remaining fields are ignored if *Version* is less than MQPMO_VERSION_3.		
OriginalMsgHandle	Original message handle	OriginalMsgHandle
NewMsgHandle	New message handle	NewMsgHandle
Action	Type of put being performed and the relationship between the original message specified by the *OriginalMsgHandle* field and the new message specified by the *NewMsgHandle* field	Action
PubLevel	Level of subscription targeted by the publication	PubLevel

Overview for MQPMO:

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQPMO structure allows the application to specify options that control how messages are placed on queues, or published to topics. The structure is an input/output parameter on the MQPUT and MQPUT1 calls.

Version: The current version of MQPMO is MQPMO_VERSION_3. Certain fields are available only in certain versions of MQPMO. If you need to port applications between several environments, you must ensure that the version of MQPMO is consistent across all environments. Fields that exist only in particular versions of the structure are identified as such in "MQPMO - Put-message options" on page 1791 and in the field descriptions.

The header, COPY, and INCLUDE files provided for the supported programming languages contain the most-recent version of MQPMO that is supported by the environment, but with the initial value of the *Version* field set to MQPMO_VERSION_1. To use fields that are not present in the version-1 structure, the application must set the *Version* field to the version number of the version required.

Character set and encoding: Data in MQPMO must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQPMO:

The MQPMO structure contains the following fields; the fields are described in **alphabetical order**:

Action (MQLONG):

This specifies the type of put being performed and the relationship between the original message specified by the OriginalMsgHandle field and the new message specified by the NewMsgHandle field. The properties of the message are chosen by the queue manager according to the value of the Action specified.

You can choose to supply the contents of the message descriptor using the MsgDesc parameter on the MQPUT or MQPUT1 calls. Alternatively it is possible not to supply the MsgDesc parameter, or to specify that it is output-only by including MQPMO_MD_FOR_OUTPUT_ONLY in the Options field of the MQPMO structure.

If the MsgDesc parameter is not supplied, or if it is specified to be output-only, then the message descriptor for the new message is populated from the message handle fields of the MQPMO, according to the rules described in this topic.

The context setting and passing activities described in Controlling context information take effect after the message descriptor has been composed.

If an incorrect action value is specified, the call fails with the reason code MQRC_ACTION_ERROR.

Any one of the following actions can be specified:

MQACTP_NEW
A new message is being put, and no relationship to a previous message is being specified by the program. The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then the queue manager generates the message descriptor using a combination of properties from OriginalMsgHandle and NewMsgHandle. Any message descriptor fields explicitly set on the new message handle take precedence over those in the original message handle.

Message data is taken from the MQPUT or MQPUT1 Buffer parameter.

MQACTP_FORWARD

A previously retrieved message is being forwarded. The original message handle specifies the message that was previously retrieved.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then the queue manager generates the message descriptor using a combination of properties from OriginalMsgHandle and NewMsgHandle. Any message descriptor fields explicitly set on the new message handle take precedence over those in the original message handle.
- If MQPMO_NEW_MSG_ID or MQPMO_NEW_CORREL_ID are specified in the MQPMO.Options, then these are honoured.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_FORWARD in the MQPD.CopyOptions
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The message data to be forwarded is taken from the MQPUT or MQPUT1 Buffer parameter.

MQACTP_REPLY

A reply is being made to a previously retrieved message. The original message handle specifies the message that was previously retrieved.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then initial message descriptor fields are chosen as follows:

Table 174. Reply message handle transformation

Field in MQMD	Value used
Report	If MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG are set: MQRO_DISCARD_MSG otherwise MQRO_NONE
MsgType	MQMT_REPLY
Expiry	If MQRO_PASS_DISCARD_AND_EXPIRY is set: Copied from the input message otherwise MQEI_UNLIMITED
Feedback	MQFB_NONE
MsgId	If MQPMO_NEW_MSG_ID is set: A new message identifier is generated else if MQRO_PASS_MSG_ID is set: Copied from the input message otherwise MQMI_NONE

CorrelId	If MQPMO_NEW_CORREL_ID is set:
	A new correlation identifier is generated
	else if MQRO_COPY_MSG_ID_TO_CORREL_ID is set:
	Copied from the MsgId field of the input message
	else if MQRO_PASS_CORREL_ID is set:
	Copied from the CorrelId field of the input message
	otherwise
	MQCI_NONE

BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Blanks
GroupId	MQGI_NONE
MsgSeqNumber	1
Offset	0
MsgFlags	MQMF_NONE
OriginalLength	MQOL_UNDEFINED

- The message descriptor is then modified by the new message handle - any message descriptor fields explicitly set as properties in the new message handle take precedence over the message descriptor fields as described above.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_REPLY in the MQPD.CopyOptions
- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The message data to be forwarded is taken from the MQPUT/MQPUT1 Buffer parameter.

MQACTP_REPORT

A report is being generated as a result of a previously retrieved message. The original message handle specifies the message causing the report to be generated.

The new message handle specifies any modifications to the properties (including any in the message descriptor) in the original message handle.

The message descriptor is composed as follows:

- If a MsgDesc is supplied on the MQPUT or MQPUT1 call, and MQPMO_MD_FOR_OUTPUT_ONLY is not in the MQPMO.Options, this is used as the message descriptor unmodified.
- If a MsgDesc is not supplied, or MQPMO_MD_FOR_OUTPUT_ONLY is in the MQPMO.Options then initial message descriptor fields are chosen as follows:

Table 175. Report message handle transformation

Field in MQMD	Value used
Report	If MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG are set: MQRO_DISCARD_MSG otherwise MQRO_NONE
MsgType	MQMT_REPORT
Expiry	If MQRO_PASS_DISCARD_AND_EXPIRY is set: Copied from the input message otherwise MQEI_UNLIMITED
MsgId	If MQPMO_NEW_MSG_ID is set: A new message identifier is generated else if MQRO_PASS_MSG_ID is set: Copied from the input message otherwise MQMI_NONE

CorrelId	If MQPMO_NEW_CORREL_ID is set:
	A new correlation identifier is generated
	else if MQRO_COPY_MSG_ID_TO_CORREL_ID is set:
	Copied from the MsgId field of the input message
	else if MQRO_PASS_CORREL_ID is set:
	Copied from the CorrelId field of the input message
	otherwise
	MQCI_NONE

BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Blanks
OriginalLength	Set to the *BufferLength*

- The message descriptor is then modified by the new message handle - any message descriptor fields explicitly set as properties in the new message handle take precedence over the message descriptor fields as described above.

The message properties are composed as follows:

- All properties from the original message handle which have MQCOPY_REPORT in the MQPD.CopyOptions

- All properties from the new message handle. For each property in the new message handle that has the same name as a property in the original message handle, the value is taken from the new message handle. The only exception to this rule is the special case when the property in the new message handle has the same name as a property in the original message handle, but the value of the property is null. In this case the property is removed from the message.

The Feedback field in the resultant MQMD represents the report that is to be generated. A Feedback value of MQFB_NONE causes the MQPUT or MQPUT1 call to fail with reason code MQRC_FEEDBACK_ERROR.

To choose the user data of the report message, WebSphere MQ consults the Report and Feedback fields in the resultant MQMD, and the Buffer and BufferLength parameters of the MQPUT or MQPUT1 call.

- If Feedback is MQFB_COA, MQFB_COD or MQFB_EXPIRATION then the value of Report is inspected.
- If any of the following cases is true, the full message data from Buffer for a length of BufferLength is used.
  - Feedback is MQFB_EXPIRATION and Report contains MQRO_EXPIRATION_WITH_FULL_DATA
  - Feedback is MQFB_COD and Report contains MQRO_COD_WITH_FULL_DATA
  - Feedback is MQFB_COA and Report contains MQRO_COA_WITH_FULL_DATA
- If any of the following cases is true, the first 100 bytes of the message (or BufferLength if this is less than 100) from Buffer are used
  - Feedback is MQFB_EXPIRATION and Report contains MQRO_EXPIRATION_WITH_DATA
  - Feedback is MQFB_COD and Report contains MQRO_COD_WITH_DATA
  - Feedback is MQFB_COA and Report contains MQRO_COA_WITH_DATA
- If Feedback is MQFB_EXPIRATION, MQFB_COD or MQFB_COA, and Report does not contain the *_WITH_FULL_DATA or *_WITH_DATA options relevant to that Feeback value, then no user data is included with the message.
- If Feedback takes a different value from those listed above, then Buffer and BufferLength are used as normal.

The derivation of the user data is shown in the following table:

Context (MQHOBJ):

If MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT is specified, this field must contain the input queue handle from which context information to be associated with the message being put is taken.

If neither MQPMO_PASS_IDENTITY_CONTEXT nor MQPMO_PASS_ALL_CONTEXT is specified, this field is ignored.

This is an input field. The initial value of this field is `0`.

InvalidDestCount (MQLONG):

This is the number of messages that could not be sent to queues in the distribution list. The count includes queues that failed to open, as well as queues that were opened successfully but for which the put operation failed. This field is also set when putting a message to a single queue that is not in a distribution list.

Note: This field is set if the *CompCode* parameter on the MQPUT or MQPUT1 call is MQCC_OK or MQCC_WARNING; it might be set if the *CompCode* parameter is MQCC_FAILED, but do not rely on this in application code.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_1.

This field is undefined on z/OS because distribution lists are not supported.

KnownDestCount (MQLONG):

This is the number of messages that the current MQPUT or MQPUT1 call has sent successfully to queues in the distribution list that are local queues. The count does not include messages sent to queues that resolve to remote queues (even though a local transmission queue is used initially to store the message). This field is also set when putting a message to a single queue that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_1.

This field is undefined on z/OS because distribution lists are not supported.

NewMsgHandle (MQHMSG):

This is an optional handle to the message being put subject to the value of the Action field. It defines the properties of the message and overrides the values of the *OriginalMsgHandle*, if specified.

On return from the **MQPUT** or **MQPUT1** call, the contents of the handle reflect the message that was actually put.

This is an input field. The initial value of this field is **MQHM_NONE**. This field is ignored if Version is less than **MQPMO_VERSION_3**.

MQPMO options (MQLONG):

The Options field controls the operation of **MQPUT** and **MQPUT1** calls.

Scope option. You can specify any or none of the MQPMO options. If more than one option is required, the values you specify for the options can be used in the following ways:

- The values can be added. Do not add the same constant more than once.
- The values can be combined using the bitwise OR operation, if the programming language supports bitwise operations.

Combinations that are not valid are noted; any other combinations are valid.

The following option controls the scope of the publications sent:

MQPMO_SCOPE_QMGR

The publication is sent only to subscribers that have subscribed on this queue manager. The publication is not forwarded to any remote publish/subscribe queue managers that have made a subscription to this queue manager, which overrides any behavior that has been set using the PUBSCOPE topic attribute.

Note: If not set, the publication scope is determined by the PUBSCOPE topic attribute.

Publishing options. The following options control the way messages are published to a topic:

MQPMO_SUPPRESS_REPLYTO

Any information specified in the *ReplyToQ* and *ReplyToQMgr* fields of the MQMD of this publication is not passed on to subscribers. If this option is used with a report option that requires a *ReplyToQ*, the call fails with MQRC_MISSING_REPLY_TO_Q.

MQPMO_RETAIN

The publication being sent is to be retained by the queue manager. This retention allows a subscriber to request a copy of this publication after the time it was published, by using the MQSUBRQ call. It also allows a publication to be sent to applications which make their subscription after the time this publication was made (unless they choose not to be sent it by using the option MQSO_NEW_PUBLICATIONS_ONLY). If an application is sent a publication which was retained, it is indicated by the MQIsRetained message property of that publication.

Only one publication can be retained at each node of the topic tree. Therefore, if there already is a retained publication for this topic, published by any other application, it is replaced with this publication. It is therefore better to avoid having more than one publisher retaining messages on the same topic.

When retained publications are requested by a subscriber, the subscription used might contain a wildcard in the topic, in which case a number of retained publications might match (at various nodes in the topic tree) and several publications might be sent to the requesting application. See the description of the "MQSUBRQ - Subscription request" on page 2093 call for more details.

For information about how retained publications interact with subscription levels, see Intercepting publications.

If this option is used and the publication cannot be retained, the message is not published and the call fails with MQRC_PUT_NOT_RETAINED.

MQPMO_NOT_OWN_SUBS

Tells the queue manager that the application does not want to send any of its publications to subscriptions it owns. Subscriptions are considered to be owned by the same application if the connection handles are the same.

MQPMO_WARN_IF_NO_SUBS_MATCHED

If no subscription matches the publication, return a completion code (*CompCode*) of MQCC_WARNING and reason code MQRC_NO_SUBS_MATCHED.

If MQRC_NO_SUBS_MATCHED is returned by the put operation, the publication was not delivered to any subscriptions. However, if the MQPMO_RETAIN option is specified on the put operation, the message is retained and delivered to any subsequently defined matching subscription.

A subscription on the topic matches the publication if any of the following conditions are met:

- The message is delivered to the subscription queue
- The message would have been delivered to the subscription queue but a problem with the queue means that the message cannot be put to the queue, and it was consequently placed on the dead letter queue or discarded.
- A routing exit is defined that suppresses delivery of the message to the subscription

A subscription on the topic does not match the publication if any of the following conditions are met:

- The subscription has a selection string that does not match the publication
- The subscription specified the MQSO_PUBLICATION_ON_REQUEST option
- The publication is not delivered because the MQPMO_NOT_OWN_SUBS option was specified on the put operation and the subscription matches the identity of the publisher

Syncpoint options. The following options relate to the participation of the MQPUT or MQPUT1 call within a unit of work:

MQPMO_SYNCPOINT

The request is to operate within the normal unit-of-work protocols. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

If MQPMO_SYNCPOINT and MQPMO_NO_SYNCPOINT are not specified, the inclusion of the put request in unit-of-work protocols is determined by the environment running the queue manager and not the environment running the application. On z/OS, the put request is within a unit of work. In all other environments, the put request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT explicitly.

Do not specify MQPMO_SYNCPOINT with MQPMO_NO_SYNCPOINT.

MQPMO_NO_SYNCPOINT
The request is to operate outside the normal unit-of-work protocols. The message is available immediately, and it cannot be deleted by backing out a unit of work.

If MQPMO_NO_SYNCPOINT and MQPMO_SYNCPOINT are not specified, the inclusion of the put request in unit-of-work protocols is determined by the environment running the queue manager and not the environment running the application. On z/OS, the put request is within a unit of work. In all other environments, the put request is not within a unit of work.

Because of these differences, an application that you want to port must not allow this option to default; specify either MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT explicitly.

Do not specify MQPMO_NO_SYNCPOINT with MQPMO_SYNCPOINT.

Message-identifier and correlation-identifier options. The following options request the queue manager to generate a new message identifier or correlation identifier:

MQPMO_NEW_MSG_ID
The queue manager replaces the contents of the *MsgId* field in MQMD with a new message identifier. This message identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

The MQPMO_NEW_MSG_ID option can also be specified when the message is being put to a distribution list; see the description of the *MsgId* field in the MQPMR structure for details.

Using this option relieves the application of the need to reset the *MsgId* field to MQMI_NONE before each MQPUT or MQPUT1 call.

MQPMO_NEW_CORREL_ID
The queue manager replaces the contents of the *CorrelId* field in MQMD with a new correlation identifier. This correlation identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

The MQPMO_NEW_CORREL_ID option can also be specified when the message is being put to a distribution list; see the description of the *CorrelId* field in the MQPMR structure for details.

MQPMO_NEW_CORREL_ID is useful in situations where the application requires a unique correlation identifier.

Group and segment options. The following options relate to the processing of messages in groups and segments of logical messages. Read the definitions that follow to help you to understand the option.

Physical message
The is the smallest unit of information that can be placed on or removed from a queue; it often corresponds to the information specified or retrieved on a single MQPUT, MQPUT1, or MQGET call. Every physical message has its own message descriptor (MQMD). Generally, physical messages are distinguished by differing values for the message identifier (*MsgId* field in MQMD), although this is not enforced by the queue manager.

Logical message
A logical message is a single unit of application information for non-z/OS platforms only. In the absence of system constraints, a logical message is the same as a physical message. But where

logical messages are extremely large, system constraints might make it advisable or necessary to split a logical message into two or more physical messages, called *segments*.

A logical message that has been segmented consists of two or more physical messages that have the same non-null group identifier (*GroupId* field in MQMD), and the same message sequence number (*MsgSeqNumber* field in MQMD). The segments are distinguished by differing values for the segment offset (*Offset* field in MQMD), which gives the offset of the data in the physical message from the start of the data in the logical message. Because each segment is a physical message, the segments in a logical message usually have differing message identifiers.

A logical message that has not been segmented, but for which segmentation has been permitted by the sending application, also has a non-null group identifier, although in this case there is only one physical message with that group identifier if the logical message does not belong to a message group. Logical messages for which segmentation has been inhibited by the sending application have a null group identifier (MQGI_NONE), unless the logical message belongs to a message group.

Message group
A message group is a set of one or more logical messages that have the same non-null group identifier. The logical messages in the group are distinguished by differing values for the message sequence number, which is an integer in the range 1 through *n*, where *n* is the number of logical messages in the group. If one or more of the logical messages is segmented, there are more than *n* physical messages in the group.

MQPMO_LOGICAL_ORDER
This option tells the queue manager how the application puts messages in groups and segments of logical messages. It can be specified only on the MQPUT call; it is not valid on the MQPUT1 call.

If MQPMO_LOGICAL_ORDER is specified, it indicates that the application uses successive MQPUT calls to:

1. Put the segments in each logical message in the order of increasing segment offset, starting from 0, with no gaps.
2. Put all the segments in one logical message before putting the segments in the next logical message.
3. Put the logical messages in each message group in the order of increasing message sequence number, starting from 1, with no gaps. IBM WebSphere MQ increments the message sequence number automatically.
4. Put all the logical messages in one message group before putting logical messages in the next message group.

For detailed information about MQPMO_LOGICAL_ORDER, see Logical and physical ordering

Context options. The following options control the processing of message context:

MQPMO_NO_CONTEXT
Both identity and origin context are set to indicate no context. This means that the context fields in MQMD are set to:

- Blanks for character fields
- Nulls for byte fields
- Zeros for numeric fields

MQPMO_DEFAULT_CONTEXT
The message is to have default context information associated with it, for both identity and origin. The queue manager sets the context fields in the message descriptor as follows:

Field in MQMD	Value used
UserIdentifier	Determined from the environment if possible; set to blanks otherwise.
AccountingToken	Determined from the environment if possible; set to MQACT_NONE otherwise.
ApplIdentityData	Set to blanks.
PutApplType	Determined from the environment.
PutApplName	Determined from the environment if possible; set to blanks otherwise.
PutDate	Set to the date when message is put.
PutTime	Set to the time when message is put.
ApplOriginData	Set to blanks.

For more information about message context, see Message context.

These are the default values and actions if no context options are specified.

MQPMO_PASS_IDENTITY_CONTEXT
The message is to have context information associated with it. Identity context is taken from the queue handle specified in the *Context* field. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT (see the preceding table for values). For more information about message context, see Message context.

For the MQPUT call, the queue must have been opened with the MQOO_PASS_IDENTITY_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_PASS_IDENTITY_CONTEXT option.

MQPMO_PASS_ALL_CONTEXT
The message is to have context information associated with it. Context is taken from the queue handle specified in the *Context* field. For more information about message context, see Controlling context information.

For the MQPUT call, the queue must have been opened with the MQOO_PASS_ALL_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_PASS_ALL_CONTEXT option.

MQPMO_SET_IDENTITY_CONTEXT
The message is to have context information associated with it. The application specifies the identity context in the MQMD structure. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT (see the preceding table for values). For more information about message context, see Message context.

For the MQPUT call, the queue must have been opened with the MQOO_SET_IDENTITY_CONTEXT option (or an option that implies it). For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_SET_IDENTITY_CONTEXT option.

MQPMO_SET_ALL_CONTEXT
The message is to have context information associated with it. The application specifies the identity, origin, and user context in the MQMD structure. For more information about message context, see Message context.

For the MQPUT call, the queue must have been opened with the MQOO_SET_ALL_CONTEXT option. For the MQPUT1 call, the same authorization check is carried out as for the MQOPEN call with the MQOO_SET_ALL_CONTEXT option.

You can specify only one of the MQPMO_*_CONTEXT context options. If you specify none, MQPMO_DEFAULT_CONTEXT is assumed.

Property options. The following option relates to the properties of the message:

MQPMO_MD_FOR_OUTPUT_ONLY
The message descriptor parameter must only be used for output to return the message descriptor of the message that was put. The message descriptor fields associated with the *NewMsgHandle*, *OriginalMsgHandle*, or both fields, of the **MQPMO** structure must be used for input.

If a valid message handle is not provided then the call fails with reason code **MQRC_MD_ERROR**.

Put response options. The following options control the response returned to an MQPUT or MQPUT1 call. You can specify only one of these options. If MQPMO_ASYNC_RESPONSE and MQPMO_SYNC_RESPONSE are not specified, MQPMO_RESPONSE_AS_Q_DEF or MQPMO_RESPONSE_AS_TOPIC_DEF is assumed.

MQPMO_ASYNC_RESPONSE
The MQPMO_ASYNC_RESPONSE option requests that an MQPUT or MQPUT1 operation is completed without the application waiting for the queue manager to complete the call. Using this option can improve messaging performance, particularly for applications using client bindings. An application can periodically check, using the MQSTAT verb, whether an error has occurred during any previous asynchronous calls.

With this option, only the following fields are guaranteed to be completed in the MQMD;

- ApplIdentityData
- PutApplType
- PutApplName
- ApplOriginData

Additionally, if either or both of MQPMO_NEW_MSG_ID or MQPMO_NEW_CORREL_ID are specified as options, the MsgId and CorrelId returned are also completed. (MQPMO_NEW_MSG_ID can be implicitly specified by specifying a blank MsgId field).

Only the preceding specified fields are completed. Other information that would normally be returned in the MQMD or MQPMO structure is undefined.

When requesting asynchronous put response for MQPUT1, the ResolvedQName and ResolvedQMgrName returned in the MQOD structure are undefined.

When requesting asynchronous put response for MQPUT or MQPUT1, a CompCode and Reason of MQCC_OK and MQRC_NONE does not necessarily mean that the message was successfully put to a queue. When developing an MQI application that uses asynchronous put response and requires confirmation that messages have been put to a queue you must check both CompCode and Reason codes from the put operations and also use MQSTAT to query asynchronous error information.

Although the success or failure of each individual MQPUT or MQPUT1 call mightnot be returned immediately, the first error that occurred under an asynchronous call can be determined later through a call to MQSTAT.

If a persistent message under syncpoint fails to be delivered using asynchronous put response, and you attempt to commit the transaction, the commit fails and the transaction is backed out with a completion code of MQCC_FAILED and a reason of MQRC_BACKED_OUT. The application can make a call to MQSTAT to determine the cause of a previous MQPUT or MQPUT1 failure.

MQPMO_SYNC_RESPONSE
Specifying this put response type ensures that the MQPUT or MQPUT1 operation is always issued synchronously. If the put operation is successful, all fields in the MQMD and MQPMO are completed.

This option ensures a synchronous response irrespective of the default put response value defined on the queue or topic object.

MQPMO_RESPONSE_AS_Q_DEF
If this value is specified for an MQPUT call, the put response type used is taken from the DEFPRESP value specified on the queue when it was first opened by the application. If a client application is connected to a queue manager at a level earlier than Version 7.0, it behaves as if MQPMO_SYNC_RESPONSE was specified.

If this option is specified for an MQPUT1 call, the value of the DEFPRESP attribute is not known before the request is sent to the server. By default, if the MQPUT1 call is using MQPMO_SYNCPOINT it behaves as for MQPMO_ASYNC_RESPONSE, and if it is using MQPMO_NO_SYNCPOINT it behaves as for MQPMO_SYNC_RESPONSE. However, you can override this default behavior by setting the Put1DefaultAlwaysSync property in the client configuration file, see CHANNELS stanza of the client configuration file.

MQPMO_RESPONSE_AS_TOPIC_DEF
MQPMO_RESPONSE_AS_TOPIC_DEF is a synonym for MQPMO_RESPONSE_AS_Q_DEF for use with topic objects.

Other options. The following options control authorization checking, what happens when the queue manager is quiescing, and resolving queue and queue manager names:

MQPMO_ALTERNATE_USER_AUTHORITY
MQPMO_ALTERNATE_USER_AUTHORITY indicates that the *AlternateUserId* field in the *ObjDesc* parameter of the MQPUT1 call contains a user identifier that is to be used to validate authority to put messages on the queue. The call can succeed only if *AlternateUserId* is authorized to open the queue with the specified options, regardless of whether the user identifier under which the application is running is authorized to do so. (This does not apply to the context options specified, however, which are always checked against the user identifier under which the application is running.)

This option is valid only with the MQPUT1 call.

MQPMO_FAIL_IF_QUIESCING
This option forces the MQPUT or MQPUT1 call to fail if the queue manager is in the quiescing state.

On z/OS, this option also forces the MQPUT or MQPUT1 call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

The call returns completion code MQCC_FAILED with reason code MQRC_Q_MGR_QUIESCING or MQRC_CONNECTION_QUIESCING.

MQPMO_RESOLVE_LOCAL_Q
Use this option to fill *ResolvedQName* in the MQPMO structure with the name of the local queue to which the message is put, and *ResolvedQMgrName* with the name of the local queue manager that hosts the local queue. For more information about MQPMO_RESOLVE_LOCAL_Q, see topic MQOO_RESOLVE_LOCAL_Q.

If you are authorized to put to a queue, you have the required authority to specify this flag on the MQPUT call; no special authority is needed.

Default option. If you need none of the options described, use the following option:

MQPMO_NONE
Use this value to indicate that no other options have been specified; all options assume their default values. MQPMO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

MQPMO_NONE is an input field. The initial value of the *Options* field is MQPMO_NONE.

OriginalMsgHandle (MQHMSG):

This is an optional handle to a message. It might have been previously retrieved from a queue. The use of this handle is subject to the value of the *Action* field; see also NewMsgHandle.

The contents of the original message handle will not be altered by the **MQPUT** or **MQPUT1** call.

This is an input field. The initial value of this field is **MQHM_NONE**. This field is ignored if Version is less than **MQPMO_VERSION_3**.

PubLevel (MQLONG):

The initial value of this field is 9. The level of subscription targeted by this publication. Only those subscriptions with the highest SubLevel less than or equal to this value receive this publication. This value must be in the range zero to 9; zero is the lowest level. However, if a publication has been retained, it is no longer available to subscribers at higher levels because it is republished at PubLevel 1.

For information, see Intercepting publications.

PutMsgRecFields (MQLONG):

This field contains flags that indicate which MQPMR fields are present in the put message records provided by the application. Use *PutMsgRecFields* only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero, or both *PutMsgRecOffset* and *PutMsgRecPtr* are zero.

For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

Use one or more of the following flags to indicate which fields are present in the put message records:

MQPMRF_MSG_ID
Message-identifier field is present.

MQPMRF_CORREL_ID
Correlation-identifier field is present.

MQPMRF_GROUP_ID
Group-identifier field is present.

MQPMRF_FEEDBACK
Feedback field is present.

MQPMRF_ACCOUNTING_TOKEN
Accounting-token field is present.

If you specify this flag, specify either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT in the *Options* field; if this condition is not satisfied, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

If no MQPMR fields are present, the following can be specified:

MQPMRF_NONE
No put-message record fields are present.

If this value is specified, either *RecsPresent* must be zero, or both *PutMsgRecOffset* and *PutMsgRecPtr* must be zero.

MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

If *PutMsgRecFields* contains flags that are not valid, or put message records are provided but *PutMsgRecFields* has the value MQPMRF_NONE, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

This is an input field. The initial value of this field is MQPMRF_NONE. This field is ignored if *Version* is less than MQPMO_VERSION_2.

PutMsgRecOffset (MQLONG):

This is the offset in bytes of the first MQPMR put message record from the start of the MQPMO structure. The offset can be positive or negative. *PutMsgRecOffset* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

When the message is being put to a distribution list, an array of one or more MQPMR put message records can be provided in order to specify certain properties of the message for each destination individually; these properties are:

- Message identifier
- Correlation identifier
- Group identifier
- Feedback value
- Accounting token

You do not need to specify all these properties, but whatever subset you choose, specify the fields in the correct order. See the description of the MQPMR structure for further details.

Usually, there must be as many put message records as there are object records specified by MQOD when the distribution list is opened; each put message record supplies the message properties for the queue identified by the corresponding object record. Queues in the distribution list that fail to open must still have put message records allocated for them at the appropriate positions in the array, although the message properties are ignored in this case.

The number of put message records can differ from the number of object records. If there are fewer put message records than object records, the message properties for the destinations that do not have put message records are taken from the corresponding fields in the message descriptor MQMD. If there are more put message records than object records, the excess are not used (although it must still be possible to access them). Put message records are optional, but if they are supplied there must be *RecsPresent* of them.

Provide the put message records in a similar way to the object records in MQOD, either by specifying an offset in *PutMsgRecOffset*, or by specifying an address in *PutMsgRecPtr*; for details of how to do this, see the *ObjectRecOffset* field described in "MQOD - Object descriptor" on page 1768.

No more than one of *PutMsgRecOffset* and *PutMsgRecPtr* can be used; the call fails with reason code MQRC_PUT_MSG_RECORDS_ERROR if both are nonzero.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

PutMsgRecPtr (MQPTR):

This is the address of the first MQPMR put message record. Use *PutMsgRecPtr* only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

You can use either *PutMsgRecPtr* or *PutMsgRecOffset* can be used to specify the put message records, but not both; see the description of the *PutMsgRecOffset* field above for details. If you do not use *PutMsgRecPtr*, set it to the null pointer or null bytes.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

RecsPresent (MQLONG):

This is the number of MQPMR put message records or MQRR response records that have been provided by the application. This number can be greater than zero only if the message is being put to a distribution list. Put message records and response records are optional; the application need not provide any records, or it can choose to provide records of only one type. However, if the application provides records of both types, it must provide *RecsPresent* records of each type.

The value of *RecsPresent* need not be the same as the number of destinations in the distribution list. If too many records are provided, the excess are not used; if too few records are provided, default values are used for the message properties for those destinations that do not have put message records (see *PutMsgRecOffset*).

If *RecsPresent* is less than zero, or is greater than zero but the message is not being put to a distribution list, the call fails with reason code MQRC_RECS_PRESENT_ERROR.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

ResolvedQMgrName (MQCHAR48):

This is the name of the destination queue manager after name resolution has been performed by the local queue manager. The name returned is the name of the queue manager that owns the queue identified by *ResolvedQName*, and can be the name of the local queue manager.

If *ResolvedQName* is a shared queue that is owned by the queue-sharing group to which the local queue manager belongs, *ResolvedQMgrName* is the name of the queue-sharing group. If the queue is owned by some other queue-sharing group, *ResolvedQName* can be the name of the queue-sharing group or the name of a queue manager that is a member of the queue-sharing group (the nature of the value returned is determined by the queue definitions that exist at the local queue manager).

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or a topic, the value returned is undefined.

This is an output field. The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResolvedQName (MQCHAR48):

This is the name of the destination queue after name resolution has been performed by the local queue manager. The name returned is the name of a queue that exists on the queue manager identified by *ResolvedQMgrName*.

A nonblank value is returned only if the object is a single queue; if the object is a distribution list or a topic, the value returned is undefined.

This is an output field. The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

ResponseRecOffset (MQLONG):

This is the offset in bytes of the first MQRR response record from the start of the MQPMO structure. The offset can be positive or negative. *ResponseRecOffset* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

When putting the message to a distribution list, you can provide an array of one or more MQRR response records to identify the queues to which the message was not sent successfully (*CompCode* field in MQRR), and the reason for each failure (*Reason* field in MQRR). The message might not have been sent either because the queue failed to open, or because the put operation failed. The queue manager sets the response records only when the outcome of the call is mixed (that is, some messages were sent successfully while others failed, or all failed but for differing reasons); reason code MQRC_MULTIPLE_REASONS from the call indicates this case. If the same reason code applies to all queues, that reason is returned in the *Reason* parameter of the MQPUT or MQPUT1 call, and the response records are not set.

Usually, there are as many response records as there are object records specified by MQOD when the distribution list is opened; when necessary, each response record is set to the completion code and reason code for the put to the queue identified by the corresponding object record. Queues in the distribution list that fail to open must still have response records allocated for them at the appropriate positions in the array, although they are set to the completion code and reason code resulting from the open operation, rather than the put operation.

The number of response records can differ from the number of object records. If there are fewer response records than object records, the application might not be able to identify all the destinations for which the put operation failed, or the reasons for the failures. If there are more response records than object records, the excess are not used (although it must still be possible to access them). Response records are optional, but if they are supplied there must be *RecsPresent* of them.

Provide the response records in a similar way to the object records in MQOD, either by specifying an offset in *ResponseRecOffset*, or by specifying an address in *ResponseRecPtr*; for details of how to do this, see the *ObjectRecOffset* field described in "MQOD - Object descriptor" on page 1768. However, use no more than one of *ResponseRecOffset* and *ResponseRecPtr*; the call fails with reason code MQRC_RESPONSE_RECORDS_ERROR if both are nonzero.

For the MQPUT1 call, this field must be zero. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is 0. This field is ignored if *Version* is less than MQPMO_VERSION_2.

ResponseRecPtr (MQPTR):

This is the address of the first MQRR response record. *ResponseRecPtr* is used only when the message is being put to a distribution list. The field is ignored if *RecsPresent* is zero.

Use either *ResponseRecPtr* or *ResponseRecOffset* to specify the response records, but not both; see the description of the *ResponseRecOffset* field above for details. If you do not use *ResponseRecPtr* set it to the null pointer or null bytes.

For the MQPUT1 call, this field must be the null pointer or null bytes. This is because the response information (if requested) is returned in the response records specified by the object descriptor MQOD.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise. This field is ignored if *Version* is less than MQPMO_VERSION_2.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length, with the initial value being the all-null byte string.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQPMO_STRUC_ID
Identifier for put-message options structure.

For the C programming language, the constant MQPMO_STRUC_ID_ARRAY is also defined; this has the same value as MQPMO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQPMO_STRUC_ID.

Timeout (MQLONG):

This is a reserved field; its value is not significant. The initial value of this field is -1.

UnknownDestCount (MQLONG):

This is the number of messages that the current MQPUT or MQPUT1 call has sent successfully to queues in the distribution list that resolve to remote queues. Messages that the queue manager retains temporarily in distribution-list form count as the number of individual destinations that those distribution lists contain. This field is also set when putting a message to a single queue that is not in a distribution list.

This is an output field. The initial value of this field is 0. This field is not set if *Version* is less than MQPMO_VERSION_1.

This field is undefined on z/OS because distribution lists are not supported.

Version (MQLONG):

Structure version number.

The value must be one of the following:

MQPMO_VERSION_1
Version-1 put-message options structure.

This version is supported in all environments.

MQPMO_VERSION_2
Version-2 put-message options structure.

This version is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

MQPMO_VERSION_3
Version-3 put-message options structure.

This version is supported in all environments.

Fields that exist only in the more-recent version of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQPMO_CURRENT_VERSION
Current version of put-message options structure.

This is always an input field. The initial value of this field is MQPMO_VERSION_1.

Initial values and language declarations for MQPMO:

Table 176. Initial values of fields in MQPMO

Field name	Name of constant	Value of constant
StrucId	MQPMO_STRUC_ID	'PMO¬'
Version	MQPMO_VERSION_1	1
Options	MQPMO_NONE	0
Timeout	None	-1
Context	None	0
KnownDestCount	None	0
UnknownDestCount	None	0
InvalidDestCount	None	0
ResolvedQName	None	Null string or blanks
ResolvedQMgrName	None	Null string or blanks
RecsPresent	None	0
PutMsgRecFields	MQPMRF_NONE	0
PutMsgRecOffset	None	0
ResponseRecOffset	None	0
PutMsgRecPtr	None	Null pointer or null bytes
ResponseRecPtr	None	Null pointer or null bytes
OriginalMsgHandle	MQHM_NONE	0
NewMsgHandle	MQHM_NONE	0
Action	MQACTP_NEW	0
PubLevel	None	9

Table 176. Initial values of fields in MQPMO (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ¬ represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 3. In the C programming language, the macro variable MQPMO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: `MQPMO MyPMO = {MQPMO_DEFAULT};`		

C declaration:

```
typedef struct tagMQPMO MQPMO;
struct tagMQPMO {
  MQCHAR4   StrucId;            /* Structure identifier */
  MQLONG    Version;            /* Structure version number */
  MQLONG    Options;            /* Options that control the action of
                                   MQPUT and MQPUT1 */
  MQLONG    Timeout;            /* Reserved */
  MQHOBJ    Context;            /* Object handle of input queue */
  MQLONG    KnownDestCount;     /* Number of messages sent
                                   successfully to local queues */
  MQLONG    UnknownDestCount;   /* Number of messages sent
                                   successfully to remote queues */
  MQLONG    InvalidDestCount;   /* Number of messages that could not
                                   be sent */
  MQCHAR48  ResolvedQName;      /* Resolved name of destination
                                   queue */
  MQCHAR48  ResolvedQMgrName;   /* Resolved name of destination queue
                                   manager */
  /* Ver:1 */
  MQLONG    RecsPresent;        /* Number of put message records or
                                   response records present */
  MQLONG    PutMsgRecFields;    /* Flags indicating which MQPMR fields
                                   are present */
  MQLONG    PutMsgRecOffset;    /* Offset of first put message record
                                   from start of MQPMO */
  MQLONG    ResponseRecOffset;  /* Offset of first response record
                                   from start of MQPMO */
  MQPTR     PutMsgRecPtr;       /* Address of first put message
                                   record */
  MQPTR     ResponseRecPtr;     /* Address of first response record */
  /* Ver:2 */
  MQHMSG    OriginalMsgHandle;  /* Original message handle */
  MQHMSG    NewMsgHandle;       /* New message handle */
  MQLONG    Action;             /* The action being performed */
  MQLONG    PubLevel;           /* Subscription level */
  /* Ver:3 */
};
```

COBOL declaration:

```
**   MQPMO structure
  10 MQPMO.
**    Structure identifier
   15 MQPMO-STRUCID           PIC X(4).
**    Structure version number
   15 MQPMO-VERSION           PIC S9(9) BINARY.
**    Options that control the action of MQPUT and MQPUT1
   15 MQPMO-OPTIONS           PIC S9(9) BINARY.
**    Reserved
   15 MQPMO-TIMEOUT           PIC S9(9) BINARY.
**    Object handle of input queue
   15 MQPMO-CONTEXT           PIC S9(9) BINARY.
**    Number of messages sent successfully to local queues
   15 MQPMO-KNOWNDESTCOUNT    PIC S9(9) BINARY.
**    Number of messages sent successfully to remote queues
   15 MQPMO-UNKNOWNDESTCOUNT  PIC S9(9) BINARY.
**    Number of messages that could not be sent
   15 MQPMO-INVALIDDESTCOUNT  PIC S9(9) BINARY.
**    Resolved name of destination queue
   15 MQPMO-RESOLVEDQNAME     PIC X(48).
**    Resolved name of destination queue manager
   15 MQPMO-RESOLVEDQMGRNAME  PIC X(48).
**    Number of put message records or response records present
   15 MQPMO-RECSPRESENT       PIC S9(9) BINARY.
**    Flags indicating which MQPMR fields are present
   15 MQPMO-PUTMSGRECFIELDS   PIC S9(9) BINARY.
**    Offset of first put message record from start of MQPMO
   15 MQPMO-PUTMSGRECOFFSET   PIC S9(9) BINARY.
**    Offset of first response record from start of MQPMO
   15 MQPMO-RESPONSERECOFFSET PIC S9(9) BINARY.
**    Address of first put message record
   15 MQPMO-PUTMSGRECPTR      POINTER.
**    Address of first response record
   15 MQPMO-RESPONSERECPTR    POINTER.
**    Original message handle
   15 MQPMO-ORIGINALMSGHANDLE PIC S9(18) BINARY.
**    New message handle
   15 MQPMO-NEWMSGHANDLE      PIC S9(18) BINARY.
**    The action being performed
   15 MQPMO-ACTION            PIC S9(9) BINARY.
**    Publish level
   15 MQPMO-PUBLEVEL          PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQPMO based,
  3 StrucId           char(4),       /* Structure identifier */
  3 Version           fixed bin(31), /* Structure version number */
  3 Options           fixed bin(31), /* Options that control the action
                                        of MQPUT and MQPUT1 */
  3 Timeout           fixed bin(31), /* Reserved */
  3 Context           fixed bin(31), /* Object handle of input queue */
  3 KnownDestCount    fixed bin(31), /* Number of messages sent
                                        successfully to local queues */
  3 UnknownDestCount  fixed bin(31), /* Number of messages sent
                                        successfully to remote queues */
  3 InvalidDestCount  fixed bin(31), /* Number of messages that could
                                        not be sent */
  3 ResolvedQName     char(48),      /* Resolved name of destination
                                        queue */
  3 ResolvedQMgrName  char(48),      /* Resolved name of destination
                                        queue manager */
  3 RecsPresent       fixed bin(31), /* Number of put message records or
                                        response records present */
  3 PutMsgRecFields   fixed bin(31), /* Flags indicating which MQPMR
```

```
                                    fields are present */
  3 PutMsgRecOffset   fixed bin(31), /* Offset of first put message
                                    record from start of MQPMO */
  3 ResponseRecOffset fixed bin(31), /* Offset of first response record
                                    from start of MQPMO */
  3 PutMsgRecPtr      pointer,       /* Address of first put message
                                    record */
  3 ResponseRecPtr    pointer,       /* Address of first response
                                    record */
  3 OriginalMsgHandle fixed bin(63), /* Original message handle */
  3 NewMsgHandle      fixed bin(63); /* New message handle */
  3 Action            fixed bin(31); /* The action being performed */
  3 PubLevel          fixed bin(31); /* Publish level */
```

High Level Assembler declaration:

```
MQPMO                   DSECT
MQPMO_STRUCID           DS   CL4   Structure identifier
MQPMO_VERSION           DS   F     Structure version number
MQPMO_OPTIONS           DS   F     Options that control the action of
*                                  MQPUT and MQPUT1
MQPMO_TIMEOUT           DS   F     Reserved
MQPMO_CONTEXT           DS   F     Object handle of input queue
MQPMO_KNOWNDESTCOUNT    DS   F     Number of messages sent successfully
*                                  to local queues
MQPMO_UNKNOWNDESTCOUNT  DS   F     Number of messages sent successfully
*                                  to remote queues
MQPMO_INVALIDDESTCOUNT  DS   F     Number of messages that could not be
*                                  sent
MQPMO_RESOLVEDQNAME     DS   CL48  Resolved name of destination queue
MQPMO_RESOLVEDQMGRNAME  DS   CL48  Resolved name of destination queue
*                                  manager
MQPMO_RECSPRESENT       DS   F     Number of put message records or
*                                  response records present
MQPMO_PUTMSGRECFIELDS   DS   F     Flags indicating which MQPMR
*                                  fields are present
MQPMO_PUTMSGRECOFFSET   DS   F     Offset of first put message record
*                                  from start of MQPMO
MQPMO_RESPONSERECOFFSET DS   F     Offset of first response record
*                                  from start of MQPMO
MQPMO_PUTMSGRECPTR      DS   F     Address of first put message
*                                  record
MQPMO_RESPONSERECPTR    DS   F     Address of first response record
MQPMO_ORIGINALMSGHANDLE DS   D     Original message handle
MQPMO_NEWMSGHANDLE      DS   D     New message handle
MQPMO_ACTION            DS   F     The action being performed
MQPMO_PUBLEVEL          DS   F     Publish level
*
MQPMO_LENGTH            EQU  *-MQPMO
                        ORG  MQPMO
MQPMO_AREA              DS   CL(MQPMO_LENGTH)
```

Visual Basic declaration:

```
Type MQPMO
  StrucId           As String*4  'Structure identifier'
  Version           As Long      'Structure version number'
  Options           As Long      'Options that control the action of'
                                 'MQPUT and MQPUT1'
  Timeout           As Long      'Reserved'
  Context           As Long      'Object handle of input queue'
  KnownDestCount    As Long      'Number of messages sent successfully'
                                 'to local queues'
  UnknownDestCount  As Long      'Number of messages sent successfully'
                                 'to remote queues'
  InvalidDestCount  As Long      'Number of messages that could not be'
                                 'sent'
  ResolvedQName     As String*48 'Resolved name of destination queue'
```

```
  ResolvedQMgrName  As String*48 'Resolved name of destination queue'
                                 'manager'
  RecsPresent       As Long      'Number of put message records or'
                                 'response records present'
  PutMsgRecFields   As Long      'Flags indicating which MQPMR fields'
                                 'are present'
  PutMsgRecOffset   As Long      'Offset of first put message record'
                                 'from start of MQPMO'
  ResponseRecOffset As Long      'Offset of first response record from'
                                 'start of MQPMO'
  PutMsgRecPtr      As MQPTR     'Address of first put message record'
  ResponseRecPtr    As MQPTR     'Address of first response record'
End Type
```

MQPMR - Put-message record:

The following table summarizes the fields in the structure.

Table 177. Fields in MQPMR

Field	Description	Topic
MsgId	Message identifier	MsgId
CorrelId	Correlation identifier	CorrelId
GroupId	Group identifier	GroupId
Feedback	Feedback or reason code	Feedback
AccountingToken	Accounting token	AccountingToken

Overview for MQPMR:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: Use the MQPMR structure to specify various message properties for a single destination when putting a message to a distribution list. MQPMR is an input/output structure for the MQPUT and MQPUT1 calls.

Character set and encoding: Data in MQPMR must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQPUT or MQPUT1 call, you can specify different values for each destination queue in a distribution list. Some of the fields are input only, others are input/output.

Note: This structure is unusual in that it does not have a fixed layout. The fields in this structure are optional, and the presence or absence of each field is indicated by the flags in the *PutMsgRecFields* field in MQPMO. Fields that are present ***must occur in the following order***:

- *MsgId*
- *CorrelId*
- *GroupId*
- *Feedback*
- *AccountingToken*

Fields that are absent occupy no space in the record.

Because MQPMR does not have a fixed layout, no definition of it is provided in the header, COPY, and INCLUDE files for the supported programming languages. The application programmer must create a declaration containing the fields that are required by the application, and set the flags in *PutMsgRecFields* to indicate the fields that are present.

Fields for MQPMR:

The MQPMR structure contains the following fields; the fields are described in **alphabetical order**:

AccountingToken (MQBYTE32):

This is the accounting token to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *AccountingToken* field in MQMD for a put to a single queue. See the description of *AccountingToken* in "MQMD - Message descriptor" on page 1705 for information about the content of this field.

If this field is not present, the value in MQMD is used.

This is an input field.

CorrelId (MQBYTE24):

This is the correlation identifier to be used for the message sent to the queue with a name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *CorrelId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *CorrelId* field.

If MQPMO_NEW_CORREL_ID is specified, a *single* new correlation identifier is generated and used for all the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that MQPMO_NEW_MSG_ID is processed (see *MsgId* field).

This is an input/output field.

Feedback (MQLONG):

This is the feedback code to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *Feedback* field in MQMD for a put to a single queue.

If this field is not present, the value in MQMD is used.

This is an input field.

GroupId (MQBYTE24):

GroupId is the group identifier to be used for the message sent to the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *GroupId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *GroupId* field. The value is processed as documented in Physical order on a queue, but with the following differences:

- GroupId is created from the QMName and a timestamp. Therefore to keep a GroupId unique keep queue manager names unique too. Also do not set the clocks back on the queue managers machine.
- In those cases where a new group identifier would be used, the queue manager generates a different group identifier for each destination (that is, no two destinations have the same group identifier).
- In those cases where the value in the field would be used, the call fails with reason code MQRC_GROUP_ID_ERROR

This is an input/output field.

MsgId (MQBYTE24):

This is the message identifier to be used for the message sent to the queue with a name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call. It is processed in the same way as the *MsgId* field in MQMD for a put to a single queue.

If this field is not present in the MQPMR record, or there are fewer MQPMR records than destinations, the value in MQMD is used for those destinations that do not have an MQPMR record containing a *MsgId* field. If that value is MQMI_NONE, a new message identifier is generated for *each* of those destinations (that is, no two of those destinations have the same message identifier).

If MQPMO_NEW_MSG_ID is specified, new message identifiers are generated for all the destinations in the distribution list, regardless of whether they have MQPMR records. This is different from the way that MQPMO_NEW_CORREL_ID is processed (see *CorrelId* field).

This is an input/output field.

Initial values and language declarations for MQPMR:

There are no initial values defined for this structure, as no structure declarations are provided in the header, COPY, and INCLUDE files for the supported programming languages. The sample declarations show how to declare the structure if all the fields are required.

C declaration:

```
typedef struct tagMQPMR MQPMR;
struct tagMQPMR {
  MQBYTE24  MsgId;            /* Message identifier */
  MQBYTE24  CorrelId;         /* Correlation identifier */
  MQBYTE24  GroupId;          /* Group identifier */
  MQLONG    Feedback;         /* Feedback or reason code */
  MQBYTE32  AccountingToken;  /* Accounting token */
};
```

COBOL declaration:

```
**   MQPMR structure
  10 MQPMR.
**    Message identifier
   15 MQPMR-MSGID           PIC X(24).
**    Correlation identifier
   15 MQPMR-CORRELID        PIC X(24).
**    Group identifier
   15 MQPMR-GROUPID         PIC X(24).
**    Feedback or reason code
   15 MQPMR-FEEDBACK        PIC S9(9) BINARY.
**    Accounting token
   15 MQPMR-ACCOUNTINGTOKEN PIC X(32).
```

PL/I declaration:

```
dcl
 1 MQPMR based,
  3 MsgId           char(24),      /* Message identifier */
  3 CorrelId        char(24),      /* Correlation identifier */
  3 GroupId         char(24),      /* Group identifier */
  3 Feedback        fixed bin(31), /* Feedback or reason code */
  3 AccountingToken char(32);      /* Accounting token */
```

Visual Basic declaration:

```
Type MQPMR
  MsgId           As MQBYTE24 'Message identifier'
  CorrelId        As MQBYTE24 'Correlation identifier'
  GroupId         As MQBYTE24 'Group identifier'
  Feedback        As Long     'Feedback or reason code'
  AccountingToken As MQBYTE32 'Accounting token'
End Type
```

MQRFH - Rules and formatting header:

This section describes the rules and formatting header, what fields it contains, and initial values of those fields.

Overview for MQRFH:

Availability: All WebSphere MQ systems, plus WebSphere MQ MQI clients connected to these systems.

Purpose: The MQRFH structure defines the layout of the rules and formatting header. Use this header to send string data in the form of name/value pairs.

Format name: MQFMT_RF_HEADER.

Character set and encoding: The fields in the MQRFH structure (including *NameValueString*) are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes the MQRFH, or by those fields in the MQMD structure if the MQRFH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Fields for MQRFH:

The MQRFH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the data that follows *NameValueString*; it does not apply to character data in the MQRFH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This specifies the numeric encoding of the data that follows *NameValueString*; it does not apply to numeric data in the MQRFH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG):

The following can be specified:

MQRFH_NONE
No flags.

The initial value of this field is MQRFH_NONE.

Format (MQCHAR8):

This specifies the format name of the data that follows *NameValueString*.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

NameValueString (MQCHARn):

This is a variable-length character string containing name/value pairs in the form:

```
name1 value1 name2 value2 name3 value3 ...
```

Each name or value must be separated from the adjacent name or value by one or more blank characters; these blanks are not significant. A name or value can contain significant blanks by prefixing and suffixing the name or value with double quotation marks; all characters between the open double quotation mark and the matching closing double quotation mark are treated as significant. In the following example, the name is `FAMOUS_WORDS`, and the value is `Hello World`:

```
FAMOUS_WORDS "Hello World"
```

A name or value can contain any characters other than the null character (which acts as a delimiter for *NameValueString*). However, to assist interoperability an application can restrict names to the following characters:

- First character: upper or lowercase alphabetic (A through Z, or a through z), or underscore.
- Subsequent characters: upper or lowercase alphabetic, decimal digit (0 through 9), underscore, hyphen, or dot.

If a name or value contains one or more double quotation marks, the name or value must be enclosed in double quotation marks, and each double quotation mark within the string must be doubled:

```
Famous_Words "The program displayed ""Hello World"""
```

Names and values are case sensitive, that is, lowercase letters are not considered to be the same as uppercase letters. For example, `FAMOUS_WORDS` and `Famous_Words` are two different names.

The length in bytes of *NameValueString* is equal to *StrucLength* minus MQRFH_STRUC_LENGTH_FIXED. To avoid problems converting the user data in some environments, make this length a multiple of four. Pad *NameValueString* with blanks to this length, or terminate it earlier by placing a null character following the last significant character in the string. The null character and the bytes following it, up to the specified length of *NameValueString*, are ignored.

Note: Because the length of this field is not fixed, the field is omitted from the declarations of the structure that are provided for the supported programming languages.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQRFH_STRUC_ID
Identifier for rules and formatting header structure.

For the C programming language, the constant MQRFH_STRUC_ID_ARRAY is also defined; this has the same value as MQRFH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQRFH_STRUC_ID.

StrucLength (MQLONG):

This is the length in bytes of the MQRFH structure, including the *NameValueString* field at the end of the structure. The length does *not* include any user data that follows the *NameValueString* field.

To avoid problems converting the user data in some environments, *StrucLength* must be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueString* field:

MQRFH_STRUC_LENGTH_FIXED
Length of fixed part of MQRFH structure.

The initial value of this field is MQRFH_STRUC_LENGTH_FIXED.

Version (MQLONG):

This is the structure version number; the value must be:

MQRFH_VERSION_1
Version-1 rules and formatting header structure.

The initial value of this field is MQRFH_VERSION_1.

Initial values and language declarations for MQRFH:

Table 178. Initial values of fields in MQRFH for MQRFH

Field name	Name of constant	Value of constant
StrucId	MQRFH_STRUC_ID	'RFH¬'
Version	MQRFH_VERSION_1	1
StrucLength	MQRFH_STRUC_LENGTH_FIXED	32
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQRFH_NONE	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQRFH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQRFH MyRFH = {MQRFH_DEFAULT};
```

C declaration:

```
typedef struct tagMQRFH MQRFH;
struct tagMQRFH {
  MQCHAR4  StrucId;         /* Structure identifier */
  MQLONG   Version;         /* Structure version number */
  MQLONG   StrucLength;     /* Total length of MQRFH including
                               NameValueString */
  MQLONG   Encoding;        /* Numeric encoding of data that follows
                               NameValueString */
  MQLONG   CodedCharSetId;  /* Character set identifier of data that
                               follows NameValueString */
  MQCHAR8  Format;          /* Format name of data that follows
                               NameValueString */
  MQLONG   Flags;           /* Flags */
};
```

COBOL declaration:

```
**   MQRFH structure
  10 MQRFH.
**    Structure identifier
   15 MQRFH-STRUCID        PIC X(4).
**    Structure version number
   15 MQRFH-VERSION        PIC S9(9) BINARY.
**    Total length of MQRFH including NAMEVALUESTRING
   15 MQRFH-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding of data that follows NAMEVALUESTRING
   15 MQRFH-ENCODING       PIC S9(9) BINARY.
**    Character set identifier of data that follows NAMEVALUESTRING
   15 MQRFH-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows NAMEVALUESTRING
   15 MQRFH-FORMAT         PIC X(8).
**    Flags
   15 MQRFH-FLAGS          PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQRFH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Total length of MQRFH including
                                     NameValueString */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows NameValueString */
  3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                                     that follows NameValueString */
  3 Format         char(8),       /* Format name of data that follows
                                     NameValueString */
  3 Flags          fixed bin(31); /* Flags */
```

High Level Assembler declaration:

```
MQRFH                 DSECT
MQRFH_STRUCID         DS   CL4  Structure identifier
MQRFH_VERSION         DS   F    Structure version number
MQRFH_STRUCLENGTH     DS   F    Total length of MQRFH including
*                               NAMEVALUESTRING
MQRFH_ENCODING        DS   F    Numeric encoding of data that follows
*                               NAMEVALUESTRING
MQRFH_CODEDCHARSETID  DS   F    Character set identifier of data that
*                               follows NAMEVALUESTRING
MQRFH_FORMAT          DS   CL8  Format name of data that follows
*                               NAMEVALUESTRING
MQRFH_FLAGS           DS   F    Flags
*
MQRFH_LENGTH          EQU  *-MQRFH
                      ORG  MQRFH
MQRFH_AREA            DS   CL(MQRFH_LENGTH)
```

Visual Basic declaration:

```
Type MQRFH
  StrucId        As String*4 'Structure identifier'
  Version        As Long     'Structure version number'
  StrucLength    As Long     'Total length of MQRFH including'
                             'NameValueString'
  Encoding       As Long     'Numeric encoding of data that follows'
                             'NameValueString'
  CodedCharSetId As Long     'Character set identifier of data that'
                             'follows NameValueString'
  Format         As String*8 'Format name of data that follows'
                             'NameValueString'
  Flags          As Long     'Flags'
End Type
```

MQRFH2 - Rules and formatting header 2:

This section describes the rules and formatting header 2, what fields it contains, and initial values of those fields.

Overview for MQRFH2:

Availability

All WebSphere MQ systems, plus WebSphere MQ MQI clients connected to these systems.

Purpose

The MQRFH2 header is based on the MQRFH header, but it allows Unicode strings to be transported without translation, and it can carry numeric data types.

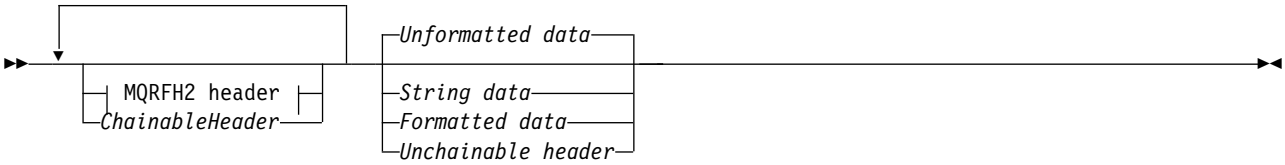
The MQRFH2 structure defines the format of the version-2 rules and formatting header. Use this header to send data that has been encoded using an XML-like syntax. A message can contain two or more MQRFH2 structures in series, with user data optionally following the last MQRFH2 structure in the series.

Format name

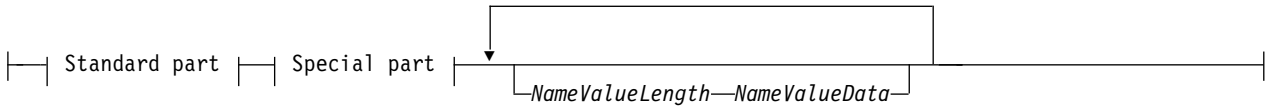
MQFMT_RF_HEADER_2

Syntax

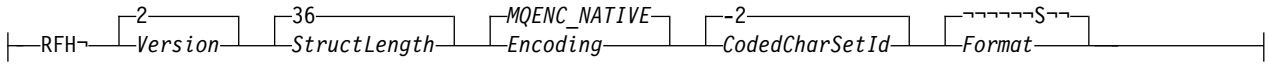
WebSphere MQ Message



MQRFH2 header:



Standard part:



Special part:

Character set and encoding

Special rules apply to the character set and encoding used for the MQRFH2 structure:

- Fields other than *NameValueData* are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes MQRFH2, or by those fields in the MQMD structure if the MQRFH2 is at the start of the application message data.

  The character set must be one that has single-byte characters for the characters that are valid in queue names.

  When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts the MQRFH2 fields, other than *NameValueData*, to the requested character set and encoding.
- *NameValueData* is in the character set given by the *NameValueCCSID* field. Only the listed Unicode character sets are valid for *NameValueCCSID* ; see the description of *NameValueCCSID* for details.

  Some character sets have a representation that depends on the encoding. If *NameValueCCSID* is one of these character sets, *NameValueData* must be in the same encoding as the other fields in the MQRFH2.

  When MQGMO_CONVERT is specified on the MQGET call, the queue manager converts *NameValueData* to the requested encoding, but does not change its character set.

Fields for MQRFH2:

The MQRFH2 structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the data that follows the last *NameValueData* field; it does not apply to character data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

The initial value of this field is MQCCSI_INHERIT.

Encoding (MQLONG):

This specifies the numeric encoding of the data that follows the last *NameValueData* field; it does not apply to numeric data in the MQRFH2 structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG):

The initial value of this field is MQRFH_NONE. MQRFH_NONE must be specified.

MQRFH_NONE
No flags.

MQRFH_INTERNAL
The MQRFH2 header contains internally set properties.

MQRFH_INTERNAL is for queue manager use.

The top 16 bits, MQRFH_FLAGS_RESTRICTED_MASK, are reserved for flags the queue manager sets. Flags that a user might set are defined in the bottom 16 bits.

Format (MQCHAR8):

This specifies the format name of the data that follows the last *NameValueData* field.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

NameValueCCSID (MQLONG):

This specifies the coded character set identifier of the data in the *NameValueData* field. This is different from the character set of the other strings in the MQRFH2 structure, and can be different from the character set of the data (if any) that follows the last *NameValueData* field at the end of the structure.

NameValueCCSID must have one of the following values:

CCSID	Meaning
1200	UCS-2 open-ended
13488	UCS-2 2.0 subset
17584	UCS-2 2.1 subset (includes the Euro symbol)
1208	UTF-8

For the UCS-2 character sets, the encoding (byte order) of the *NameValueData* must be the same as the encoding of the other fields in the MQRFH2 structure. Surrogate characters (X'D800' through X'DFFF') are not supported.

Note: If *NameValueCCSID* does not have one of the values listed above, and the MQRFH2 structure requires conversion on the MQGET call, the call completes with reason code MQRC_SOURCE_CCSID_ERROR and the message is returned unconverted.

The initial value of this field is 1208.

NameValueData (MQCHARn):

NameValueData is a variable length field that contains a folder containing name/value pairs or message properties. A folder is a variable-length character string containing data encoded using an XML-like syntax. The length in bytes of the character string is given by the *NameValueLength* field that precedes the *NameValueData* field. The length must be a multiple of four.

The *NameValueLength* and *NameValueData* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

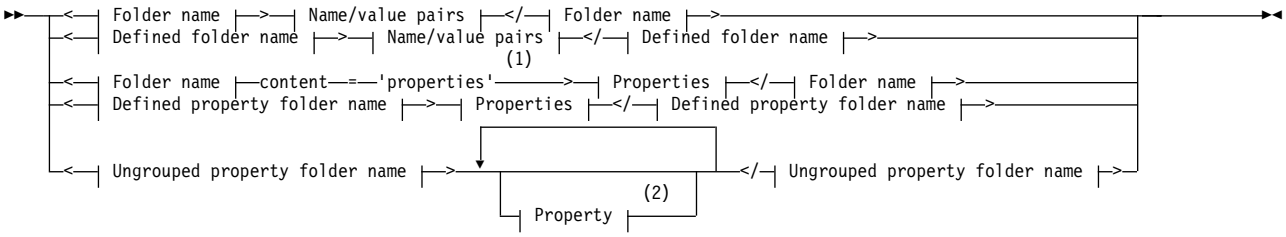
```
length1 data1 length2 data2 length3 data3
```

NameValueData is *not* converted to the character set specified on the MQGET call. Even if the message is retrieved with the MQGMO_CONVERT option in effect *NameValueData* remains in its original character set. However, *NameValueData is* converted to the encoding specified on the MQGET call.

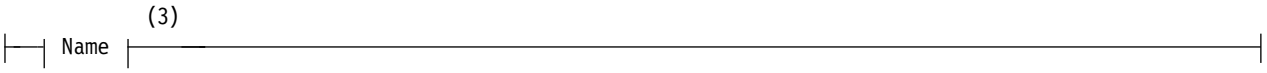
Note: Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

: The terms “defined” and “reserved” are used in the syntax diagram. “Defined” means that the name is used by WebSphere MQ. “Reserved” means that the name is reserved for future use by WebSphere MQ.

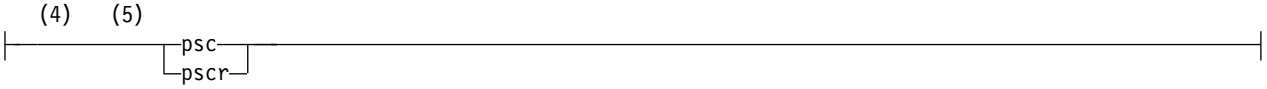
NameValueData **syntax**



Folder name:



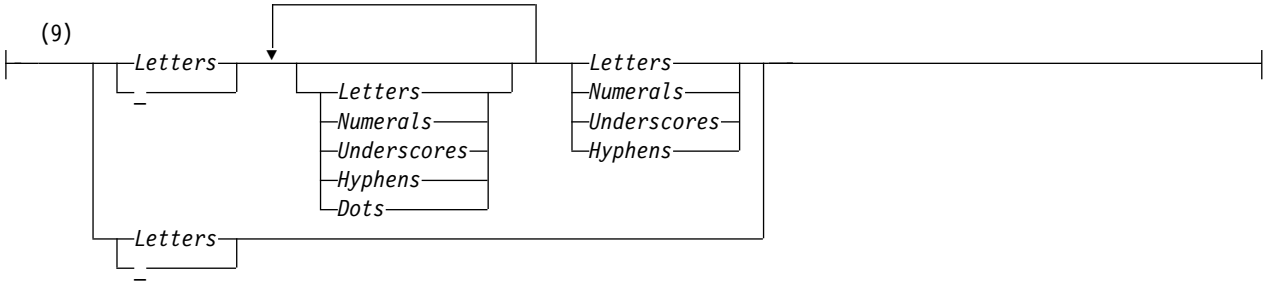
Defined folder name:



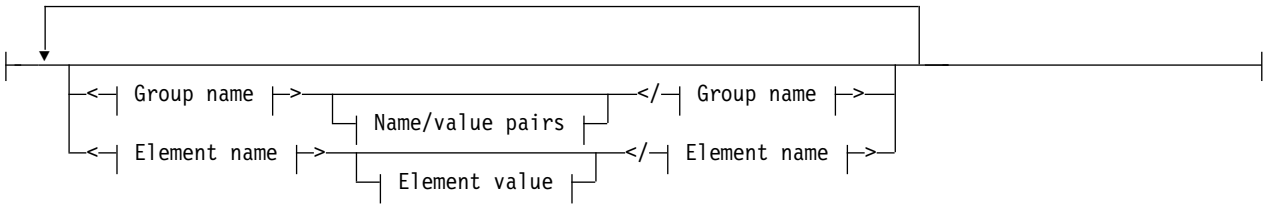
Defined property folder name:

Ungrouped property folder name:

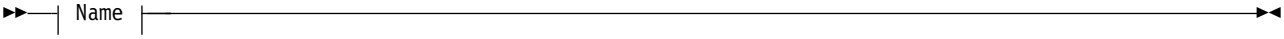
Name:



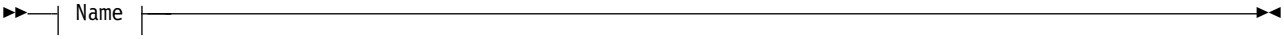
Name/value pairs:



Group name:

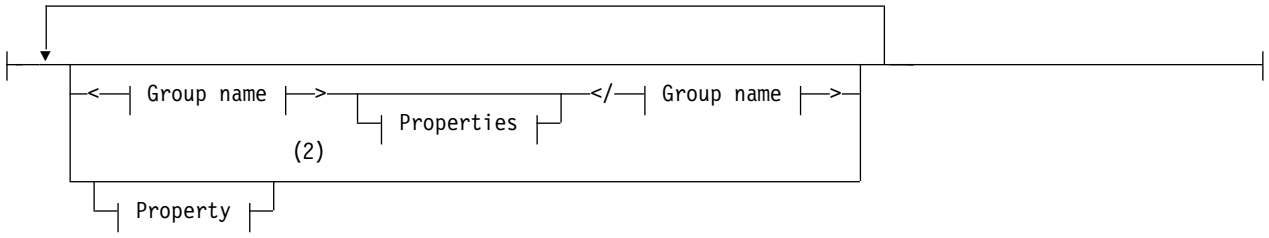


Element name:

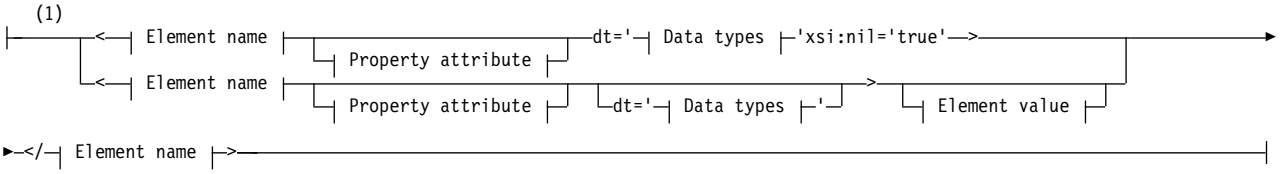


Element value:

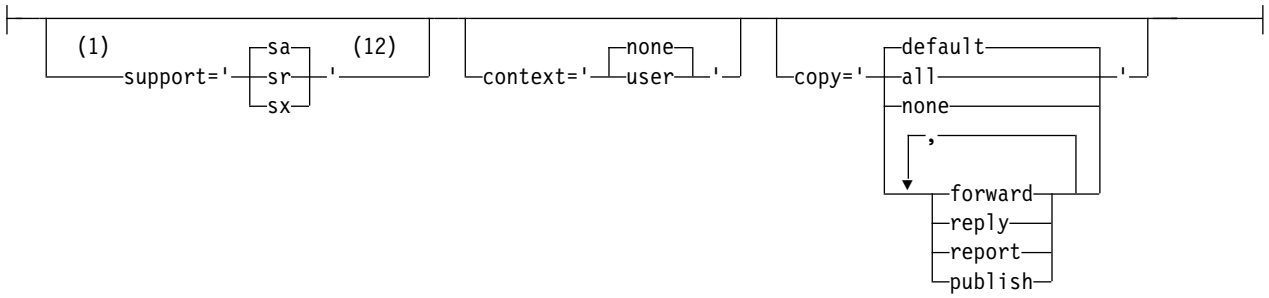
Properties:



Property:



Property attribute:



Data types:

Notes:

1 Double quotation marks or single quotation marks are valid.

2 Do not use an invalid property name; see “Invalid property name” on page 1837. Use a reserved property name only for its defined purpose; see “Defined property names” on page 1837.

3 Do not use an invalid or reserved folder name; see “Invalid path name” on page 1836 and “Reserved folder or property folder name” on page 1836. Use a defined folder name only for its defined purpose; see “Defined folder name” on page 1828.

4 The name must be in lowercase.

5 Only one **psc** and **pscr** folder is supported.

6 Only properties in the first MQRFH2 header are significant. WebSphere Application Server Service Integration Bus ignores sib, sib_context, and sib_usr folders in subsequent MQRFH2 headers.

7 Not more than one **usr** folder must be present in an MQRFH2. Properties in the **usr** folder must occur no more than once.

8 Only properties in the first **mq** folder are significant. If the folder is UTF-8, only single byte UTF-8 characters are supported. The only white space character is Unicode U+0020.

9 Valid characters are defined in the W3C XML specification, and consist essentially of Unicode categories Ll, Lu, Lo, Lt, Nl, Mc, Mn, Lm, and Nd.

10 All characters are significant. Leading and trailing blanks are part of the element value.

11 Do not use an invalid character; see "Invalid characters" on page 1836. Use an escape sequence, rather than these invalid characters.

12 The support property attribute is only valid on the **mq** folder

Folder name

NameValueData contains a single folder. To create multiple folders, create multiple *NameValueData* fields. You can create multiple *NameValueData* fields in a single MQRFH2 header within a message. Alternatively you can create multiple chained MQRFH2 headers, each containing multiple *NameValueData* fields.

The order of MQRFH2 headers, and the order of *NameValueData* fields makes no difference to the logical contents of a folder. If the same folder is present more than once in a message the folder is parsed as a whole. If the same property occurs in muliple instances of the same folder, it is parsed as a list.

A correct parse of an MQRFH2 is not affected by the alternative ways a folder can be physically stored in a message.

Four folders do not follow this rule. Only the first instance of the mq, sib, sib_context, and sib_usr folder are parsed.

If the same property occurs more than once in the combined contents of the chained MQRFH2 headers, only the first instance of the property is parsed. If a property is set using an API call, such as MQSETMP, and added to an MQRFH2 directly by an application, the API call takes precedence.

A folder name is the name of a folder containing name/value pairs or groups. Groups and name/value pairs can be mixed at the same level in the folder tree; see Figure 10. Do not combine a group name and an element name; see Figure 11

```
<group1><nvp1>value</nvp1></group1><group2><nvp2>value</nvp2></group2>
<group3><nvp1>value</nvp1></group3><nvp3>value</nvp3>
```

Figure 10. Correct uses of groups and name/value pairs

```
<group1><nvp1>value</nvp1>value</group1>
```

Figure 11. Incorrect use of groups and name/value pairs

Do not use an invalid or reserved folder name; see “Invalid path name” on page 1836 and “Reserved folder or property folder name” on page 1836. Use a defined folder name only for its defined purpose; see “Defined folder name.”

If you add the attribute `'content=properties'` to the folder name tag, the folder becomes a property folder; see Figure 12.

```
<myFolder></myfolder>
<myPropertyFolder contents='properties'></myPropertyFolder>
```

Figure 12. Example of a folder and a property folder

Folder names are case-sensitive. Folder names and property folder names share the same namespace. They must have different names. `Folder1` in Figure 13 must be a different name to `Folder2` in Figure 14.

```
<Folder1><NVP1>value</NVP1></Folder1>
```

Figure 13. `Folder1` namespace

```
<Folder2 content='properties'><Property1>value</Property1></Folder2>
```

Figure 14. `Folder2` namespace

Groups, properties, and name/value pairs in different folders have different namespaces. `Property1` in Figure 14 is a different property to `Property1` in Figure 15.

```
<Folder3 content='properties'><Property1>value</Property1></Folder3>
```

Figure 15. `Folder3` namespace

Property folders are different to non-property folders in two important respects:

1. Property folders contain properties, and non-property folders contain name/value pairs. The folders differ slightly, syntactically.
2. Use the defined interfaces, such as the properties MQI, or JMS message properties, to access message properties. The interfaces ensure the property folders in the `MQRFH2` are well-formed. A well-formed property folder is interoperable between queue managers on different platforms and different releases.

The message property MQI is a robust way to read and write an `MQRFH2`, and avoids the difficulties of parsing an `MQRFH2` correctly.

Defined folder name

A defined folder name is the name of a folder that is reserved for use by WebSphere MQ, or another product. Do not create a folder of the same name, and do not add your own name/value pairs to the folders. The defined folders are **psc** and **pscr**.

psc and **pscr** are used by queued publish/subscribe.

A segmented message put with either MQMF_SEGMENT or MQMF_SEGMENTATION_ALLOWED cannot contain an MQRFH2 with a defined folder name. The MQPUT fails with reason code 2443, MQRC_SEGMENTATION_NOT_ALLOWED.

Defined property folder name

A defined property folder name is the name of a property folder that is used by WebSphere MQ, or another product. For the names of the folders and their contents, see Property folders. Defined property folder names are a subset of all the folder names reserved by WebSphere MQ; see "Reserved folder or property folder name" on page 1836.

Any element stored in a defined property folder is a property. An element stored in a defined property folder must not have a content='properties' attribute.

You can add properties only to the defined property folders **usr**, **mq_usr**, and **sib_usr**. In other property folders, such as **mq** and **sib**, WebSphere MQ ignores or throws away properties it does not recognize.

The description of each defined property folder lists the properties that WebSphere MQ has defined that can be used by application programs. Some of the properties are accessed indirectly by setting or getting a JMS property, and some are accessed directly using the MQSETMP and MQINQMP MQI calls.

The defined property folders also contain other properties that WebSphere MQ has reserved, but which applications do not have access to. The names of the reserved properties are not listed. No reserved properties are present in the **usr**, **mq_usr**, and **sib_usr** property folders. But do not create properties with invalid property names; see "Invalid property name" on page 1837.

Property folders

jms

jms contains JMS header fields, and JMSX properties that cannot be fully expressed in the MQMD. The **jms** folder is always present in a JMS MQRFH2.

Table 179. jms property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
JMSDestination	jms.Dst	string	<jms><Dst>*destination*</Dst></jms>
JMSExpiration	jms.Exp	i8	<jms><Exp>*expiration*</Exp></jms>
JMSCorrelation	jms.Cid	string	<jms><Cid>*correlationId*</Cid></jms>
JMSDelivery	jms.Dlv	i4	<jms><Dlv>*delivery*</Dlv></jms>
JMSPriority	jms.Pri	i4	<jms><Pri>*priority*</Pri></jms>
JMSReplyTo	jms.Rto	string	<jms><Rto>*replyToURI*</Rto></jms>
JMSTimeStamp	jms.Tms	i8	<jms><Tms>*timestamp*</Tms></jms>
JMSXGroupID	jms.Gid	string	<jms><Gid>*groupId*</Gid></jms>
JMSXGroupSeq	jms.Seq	i4	<jms><Seq>*messageSequenceNo*</Seq></jms>

Do not add your own properties in the jms folder.

mcd

mcd contains properties that describe the format of the message. For example, the message service domain Msd property identifies a JMS message as being JMSTextMessage, JMSBytesMessage, JMSStreamMessage, JMSMapMessage, JMSObjectMessage, or null.

The mcd folder is always present in a JMS message containing an MQRFH2.

It is always present in a message containing an MQRFH2 sent from WebSphere Message Broker. It describes the domain, format, type, and message set of a message.

Table 180. mcd property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
	mcd.Msd	string	<mcd><Msd>*messageDomain*</Msd></mcd>
	mcd.Set	string	<mcd><Set>*messageDomain*</Set></mcd>
	mcd.Type	string	<mcd><Type>*messageDomain*</Type></mcd>
	mcd.Fmt	string	<mcd><Fmt>*messageDomain*</Fmt></mcd>

Do not add your own properties in the mcd folder.

mq_usr

mq_usr contains application-defined properties that are not exposed as JMS user-defined properties. Properties that do not meet JMS requirements can be placed in this folder.

You can create properties in the mq_usr folder. Properties you create in the mq_usr are like properties you create in new folders with the content='properties' attribute.

sib

sib contains WebSphere Application Server service integration bus (WAS/SIB) system message properties. sib properties are not exposed as JMS properties to WebSphere MQ JMS applications because they are not of the supported types. For example, some sib properties cannot be exposed as JMS properties because they are byte arrays. Some sib properties are exposed to WAS/SIB applications as JMS_IBM_* properties; these include forward and reverse routing paths properties.

Do not add your own properties in the sib folder.

sib_context

sib_context contains WAS/SIB system message properties that are not exposed to WAS/SIB user applications or as JMS properties. sib_context contains security and transactional properties that are used for web services.

Do not add your own properties in the sib_context folder.

sib_usr

sib_usr contains WAS/SIB user message properties that are not exposed as JMS user properties because they are not of supported types. sib_usr is exposed to WAS/SIB applications in the SIMessage interface; see Developing Service Integration.

The type of a sib_usr property must be bin.hex, and the value must be in the correct format. If a WebSphere MQ application writes a bin.hex typed element to the folder in the wrong format, the application receives an IOException. If the data type of the property is not bin.hex the application receives a ClassCastException.

Do not attempt to make JMS user properties available to WAS/SIB by using this folder; instead use the usr folder.

You can create properties in the sib_usr folder.

usr

usr contains application-defined JMS properties associated with the message. The usr folder is present only if an application has set an application-defined property.

usr is the default property folder. If a property is set without a folder name, it is placed in the usr folder.

Table 181. usr property name, synonym, data type, and folder.

The web services property values are described in MQRFH2 SOAP settings

Property synonym	Property name	Data type	Folder
	usr.contentType	string	<usr><contentType>text/xml; charset=utf-8</contentType></usr>
	usr.endPointURL	string	<usr><endPointURL>*URI*</endPointURL></usr>
	usr.targetService	string	<usr><targetService>*serviceName*</targetService></usr>
	usr.soapAction	string	<usr><soapAction>*name*</soapAction></usr>
	usr.transportVersion	string	<usr><transportVersion>*version*</transportVersion></usr>

You can create properties in the usr folder.

A segmented message put with either MQMF_SEGMENT or MQMF_SEGMENTATION_ALLOWED cannot contain an MQRFH2 with a defined property folder name. The MQPUT fails with reason code 2443, MQRC_SEGMENTATION_NOT_ALLOWED.

Ungrouped property folder name

ibm

ibm contains properties that are used only by WebSphere MQ.

Table 182. ibm property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
	ibm.rfp	string	<ibm><rfp>fingerprint</rfp></ibm>

Do not add your own properties in the ibm folder.

mq

mq contains properties that are used only by WebSphere MQ.

The following restrictions apply to properties in the mq folder:

- Only properties in the first significant mq folder in the message are acted upon by MQ; properties in any other mq folder in the message are ignored.
- Only single-byte UTF-8 characters are allowed in the folder. A multi-byte character in the folder, can cause parsing to fail, and the message to be rejected.
- Do not use escape strings in the folder. An escape string is treated as the actual value of the element.
- Only Unicode character U+0020 is treated as white space within the folder. All other characters are treated as significant and can cause parsing of the folder to fail, and the message to be rejected.

If parsing of the mq folder fails, or if the folder does not observe these restrictions, the message is rejected with reason code 2527, MQRC_RFH_RESTRICTED_FORMAT_ERR.

Do not add your own properties in the mq folder.

mqema

mqema contains properties that are used only by WebSphere Application Server. The folder has been replaced by mqext.

Do not add your own properties in the mqema folder.

mqext

mqext contains properties that are used only by WebSphere Application Server. The folder is present only if the application has set at least one of the IBM defined properties.

Table 183. mqext property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
JMSArmCorrelator	mqext.Arm	string	<mqext><Arm>*armCorrelator*</Arm></mqext>
JMSRMCorrelator	mqext.Wrm	string	<mqext><Wrm>*wrmCorrelator*</Wrm></mqext>

Do not add your own properties in the mqext folder.

mqps

mqps contains properties that are used only by WebSphere MQ publish/subscribe. The folder is present only if the application has set at least one of the integrated publish/subscribe properties.

Table 184. mqps property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
MQTopicString	mqps.Top	string	<mqps><Top>*topicString*</Top></mqps>
MQSubUserData	mqps.Sud	string	<mqps><Sud>*subscriberUserData*</Sud></mqps>
MQIsRetained	mqps.Ret	boolean	<mqps><Ret>*isRetained*</Ret></mqps>
MQPubOptions	mqps.Pub	i8	<mqps><Pub>*publicationOptions*</Pub></mqps>
MQPubLevel	mqps.Pbl	i8	<mqps><Pbl>*publicationLevel*</Pbl></mqps>
MQPubTime	mqpse.Pts	string	<mqps><Pts>*publicationTime*</Pts></mqps>
MQPubSeqNum	mqpse.Seq	i8	<mqps><Seq>*publicationSequenceNumber*</Seq></mqps>
MQPubStrIntData	mqpse.Sid	string	<mqps><Sid>*publicationData*</Sid></mqps>
MQPubFormat	mqpse.Pfmt	i8	<mqps><Pfmt>*messageFormat*</Pfmt></mqps>

Do not add your own properties in the mqps folder.

mq_svc

mq_svc contains properties used by SupportPac MA93.

Do not add your own properties in the mq_svc folder.

mqtt

mqtt contains properties use by WebSphere MQ Telemetry

Table 185. mqtt property name, synonym, data type, and folder

Property synonym	Property name	Data type	Folder
	mqtt.clientId	string	<mqtt><clientId>*topicString*</clientId></mqtt>
	mqtt.qos	i4	<mqtt><qos>*qualityOfService*</qos></mqtt>
	mqtt.msgid	string	<mqtt><msgid>*messageIdentifier*</msgid></mqtt>

Do not add your own properties in the mqtt folder.

A segmented message put with either MQMF_SEGMENT or MQMF_SEGMENTATION_ALLOWED cannot contain an MQRFH2 with an ungrouped property folder name. The MQPUT fails with reason code 2443, MQRC_SEGMENTATION_NOT_ALLOWED.

Name/value pairs

In the syntax diagram, “Name/value pairs” describes the content of an ordinary folder. An ordinary folder contains groups, and elements. An element is a name/value pair. A group contains elements and other groups.

In terms of trees, elements are leaf nodes, and groups are internal nodes. An internal node, and the folder, which is the root node, can contain a mixture of internal nodes and leaf nodes. A node cannot be both an internal node and a leaf node at the same time; see Figure 11 on page 1827.

Properties

In the syntax diagram, “Properties” describes the content of a property folder. A property folder contains groups, and properties. A property is a name/value pair with an optional data type attribute. A group contains properties and other groups.

In terms of trees, properties are leaf nodes, and groups are internal nodes. An internal node, and the property folder, which is the root node, can contain a mixture of internal nodes and leaf nodes. A node cannot be both an internal node and a leaf node at the same time; see Figure 11 on page 1827.

Property

A message property is a name/value pair in a property folder. It can optionally include a data type attribute and a property attribute; for an example, see Figure 16. If the data type attribute is omitted, the property type is string.

```
<pf><p1 dt='i8' >value</p1></pf>
```

Figure 16. Data type attribute

The name of a message property is its full path name, with the XML-like, <> syntax, replaced by dots. For example, `myPropertyFolder1.myGroup1.myGroup2.myProperty1` is mapped to a *NameValueData* string in Figure 17. The string is formatted for easier reading.

```
<myPropertyFolder1>
    <myGroup1>
        <myGroup2>
            <myProperty1>value</myProperty1>
        </myGroup2>
    </myGroup1>
</myPropertyFolder1>
```

Figure 17. Single property name mapping

A property folder can contain multiple properties. For example the properties in Figure 18 on page 1834 are mapped to the property folder in Figure 19 on page 1834

```
myPropertyFolder1.myProperty4
myPropertyFolder1.myGroup1.myGroup2.myProperty1
myPropertyFolder1.myGroup1.myGroup2.myProperty2
myPropertyFolder1.myGroup1.myProperty3
```

Figure 18. Multiple properties with the same root name

```
<myPropertyFolder1>
    <myProperty4>value</myProperty4>
    <myGroup1>
        <myGroup2>
            <myProperty1>value</myProperty1>
            <myProperty2>value</myProperty2>
        </myGroup2>
        <myProperty3>value</myProperty3>
    </myGroup1>
</myPropertyFolder1>
```

Figure 19. Multiple property name mapping

Name

A name must begin with a *Letter* or an *Underscore*. It must not contain a *Colon*, not end in a *Period* and contain only *Letters*, *Numerals*, *Underscores*, *Hyphens*, and *Dots*. Valid characters are defined in the W3C XML specification, and consist essentially of Unicode categories `Ll`, `Lu`, `Lo`, `Lt`, `Nl`, `Mc`, `Mn`, `Lm`, and `Nd`.

The complete path of a property or name/value pair must not break the rule described in "Invalid path name" on page 1836. Paths are restricted to 4095 bytes, must not contain Unicode compatibility characters, and must not start with the string `XML`.

Group name

A group name has the same syntax as a name. Group names are optional. Properties and name/value pairs can be placed in the root of a folder. Use groups if it helps to organize properties and name/value pairs.

Element name

An element name has the same syntax as a name.

Element value

An element value includes all the white space between the <*Element name*> tag and the </*Element name*>. Do not use the two characters < and & in a value. Replace then with < and &.

Property attribute

The property attributes map property descriptor fields: The mappings are as follows:

Support

sa MQPD_SUPPORT_OPTIONAL

sr MQPD_SUPPORT_REQUIRED

sx MQPD_SUPPORT_REQUIRED_IF_LOCAL

Context

none
MQPD_NO_CONTEXT

user
MQPD_USER _CONTEXT

CopyOptions

forward
MQPD_COPY_FORWARD

reply
MQPD_COPY_REPLY

report
MQPD_COPY_REPORT

publish
MQPD_COPY_PUBLISH

all
MQPD_COPY_ALL

Do not use **all** in combination with other options.

default
MQPD_COPY_DEFAULT

Do not use **default** in combination with other options. **default** is the same as **forward** + **report** + **publish**

none
MQPD_COPY_NONE

Do not use **none** in combination with other options.

The Support property attributes are only applicable to properties in the **mq** folder.

The Context and CopyOptions property attributes are applicable to all property folders.

Data type

MQRFH2 data types map to message property types as follows:

Table 186. Data type mappings

MQRFH2 data type	**Message property type**
bin.hex	MQBYTE[]
boolean	MQBOOL
i1	MQINT8
i2	MQINT16
i4	MQINT32
i8	MQINT64
r4	MQFLOAT32
r8	MQFLOAT64
string	MQCHAR[]

Any element without a data type is assumed to be of type string.

A null value is indicated by the element attribute `xsi:nil='true'`. Do not use the attribute `xsi:nil='false'` for non-null values. For example, the following property has a null value:

```
<NullProperty
xsi:nil='true'></NullProperty>
```

A byte or character string property can have an empty value. An empty value is represented by an `MQRFH2` element with a zero length element value. For example, the following property has an empty value:

```
<EmptyProperty></EmptyProperty>
```

Reserved folder or property folder name

Restrict the name of a folder or property folder not to start with any of the following strings. The prefixes are reserved for folder or property names created by IBM.

Notes:

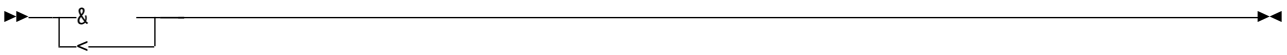
1 A reserved folder or property name contains any mixture of lower and uppercase letters.

Invalid path name

Restrict the complete path of a name/value pair or a property not to include any of the following strings.

Invalid characters

Always use the escape sequences `&` and `<` instead of the literals “&” and “<”.

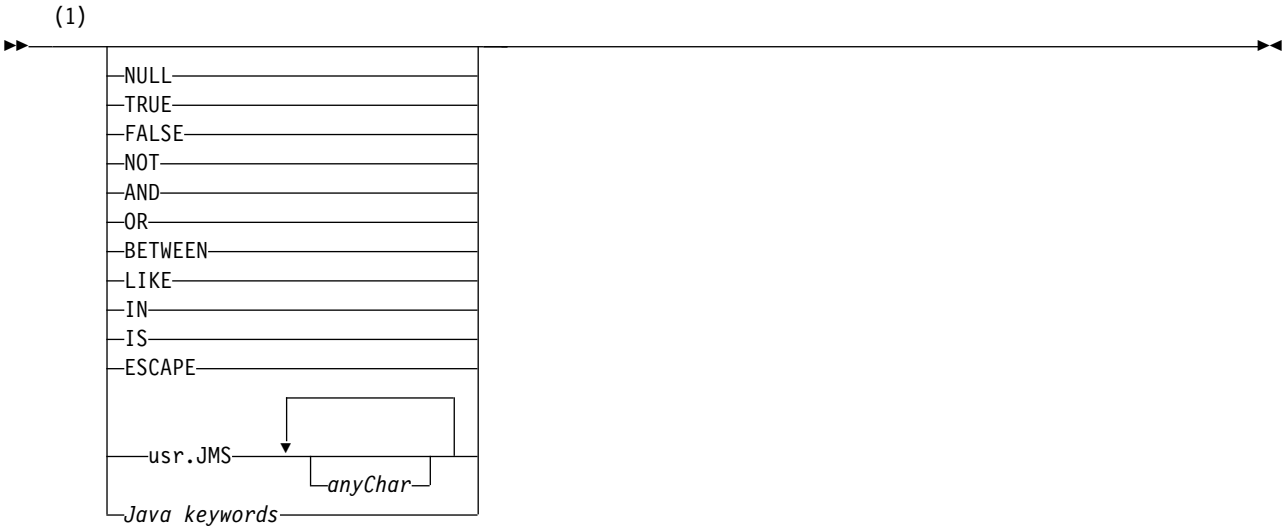


Defined property names

Defined property names are the names of properties that are defined by WebSphere MQ, or other products, and used by WebSphere MQ and user applications. Defined properties exist only in defined property folders. Defined property names are described in the description of property folders; see Property folders.

Invalid property name

Do not construct property names that match the following rule. The rule applies to the full property path that names a property, and not only to the property element name.



Notes:

1 An invalid property name can contain any combination of upper and lowercase.

NameValueLength (MQLONG):

The length of the corresponding NameValueData field

This specifies the length in bytes of the data in the *NameValueData* field. *NameValueLength* must be a multiple of four.

Note: The *NameValueLength* and *NameValueData* fields are optional, but if present they must occur as a pair and be adjacent. The pair of fields can be repeated as many times as required, for example:

```
length1 data1 length2 data2 length3 data3
```

Because these fields are optional, they are omitted from the declarations of the structure that are provided for the various programming languages supported.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQRFH_STRUC_ID
Identifier for rules and formatting header structure.

For the C programming language, the constant MQRFH_STRUC_ID_ARRAY is also defined; this has the same value as MQRFH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQRFH_STRUC_ID.

StrucLength (MQLONG):

This is the length in bytes of the MQRFH2 structure, including the *NameValueLength* and *NameValueData* fields at the end of the structure. It is valid for there to be multiple pairs of *NameValueLength* and *NameValueData* fields at the end of the structure, in the sequence:

```
length1, data1, length2, data2, ...
```

StrucLength does *not* include any user data that might follow the last *NameValueData* field at the end of the structure.

To avoid problems with converting the user data in some environments, *StrucLength* must be a multiple of four.

The following constant gives the length of the *fixed* part of the structure, that is, the length excluding the *NameValueLength* and *NameValueData* fields:

MQRFH_STRUC_LENGTH_FIXED_2
Length of fixed part of MQRFH2 structure.

The initial value of this field is MQRFH_STRUC_LENGTH_FIXED_2.

Version (MQLONG):

This is the structure version number; the value must be:

MQRFH_VERSION_2
Version-2 rules and formatting header structure.

The initial value of this field is MQRFH_VERSION_2.

Initial values and language declarations for MQRFH2:

Table 187. Initial values of fields in MQRFH2 for MQRFH2

Field name	Name of constant	Value of constant
StrucId	MQRFH_STRUC_ID	'RFH¬'
Version	MQRFH_VERSION_2	2
StrucLength	MQRFH_STRUC_LENGTH_FIXED_2	36
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_INHERIT	-2
Format	MQFMT_NONE	Blanks
Flags	MQRFH_NONE	0
NameValueCCSID	None	1208

Table 187. Initial values of fields in MQRFH2 for MQRFH2 (continued)

Field name	Name of constant	Value of constant
Notes: 1. The symbol ¬ represents a single blank character. 2. In the C programming language, the macro variable MQRFH2_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: `MQRFH2 MyRFH2 = {MQRFH2_DEFAULT};`		

C declaration:

```
typedef struct tagMQRFH2 MQRFH2;
struct tagMQRFH2 {
  MQCHAR4  StrucId;         /* Structure identifier */
  MQLONG   Version;         /* Structure version number */
  MQLONG   StrucLength;     /* Total length of MQRFH2 including all
                               NameValueLength and NameValueData
                               fields */
  MQLONG   Encoding;        /* Numeric encoding of data that follows
                               last NameValueData field */
  MQLONG   CodedCharSetId;  /* Character set identifier of data that
                               follows last NameValueData field */
  MQCHAR8  Format;          /* Format name of data that follows last
                               NameValueData field */
  MQLONG   Flags;           /* Flags */
  MQLONG   NameValueCCSID;  /* Character set identifier of
                               NameValueData */
};
```

COBOL declaration:

```
**   MQRFH2 structure
  10 MQRFH2.
**    Structure identifier
   15 MQRFH2-STRUCID        PIC X(4).
**    Structure version number
   15 MQRFH2-VERSION        PIC S9(9) BINARY.
**    Total length of MQRFH2 including all NAMEVALUELENGTH and
**    NAMEVALUEDATA fields
   15 MQRFH2-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding of data that follows last NAMEVALUEDATA field
   15 MQRFH2-ENCODING       PIC S9(9) BINARY.
**    Character set identifier of data that follows last NAMEVALUEDATA
**    field
   15 MQRFH2-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows last NAMEVALUEDATA field
   15 MQRFH2-FORMAT         PIC X(8).
**    Flags
   15 MQRFH2-FLAGS          PIC S9(9) BINARY.
**    Character set identifier of NAMEVALUEDATA
   15 MQRFH2-NAMEVALUECCSID PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQRFH2 based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Total length of MQRFH2 including
                                     all NameValueLength and
                                     NameValueData fields */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows last NameValueData field */
  3 CodedCharSetId fixed bin(31), /* Character set identifier of data
                                     that follows last NameValueData
                                     field */
  3 Format         char(8),       /* Format name of data that follows
                                     last NameValueData field */
  3 Flags          fixed bin(31), /* Flags */
  3 NameValueCCSID fixed bin(31); /* Character set identifier of
                                     NameValueData */
```

High Level Assembler declaration:

```
MQRFH                 DSECT
MQRFH_STRUCID         DS   CL4  Structure identifier
MQRFH_VERSION         DS   F    Structure version number
MQRFH_STRUCLENGTH     DS   F    Total length of MQRFH2 including all
*                               NAMEVALUELENGTH and NAMEVALUEDATA fields
MQRFH_ENCODING        DS   F    Numeric encoding of data that follows
*                               last NAMEVALUEDATA field
MQRFH_CODEDCHARSETID  DS   F    Character set identifier of data that
*                               follows last NAMEVALUEDATA field
MQRFH_FORMAT          DS   CL8  Format name of data that follows last
*                               NAMEVALUEDATA field
MQRFH_FLAGS           DS   F    Flags
MQRFH_NAMEVALUECCSID  DS   F    Character set identifier of
*                               NAMEVALUEDATA
*
MQRFH_LENGTH          EQU  *-MQRFH
                      ORG  MQRFH
MQRFH_AREA            DS   CL(MQRFH_LENGTH)
```

Visual Basic declaration:

```
Type MQRFH2
  StrucId        As String*4 'Structure identifier'
  Version        As Long     'Structure version number'
  StrucLength    As Long     'Total length of MQRFH2 including all'
                             'NameValueLength and NameValueData fields'
  Encoding       As Long     'Numeric encoding of data that follows'
                             'last NameValueData field'
  CodedCharSetId As Long     'Character set identifier of data that'
                             'follows last NameValueData field'
  Format         As String*8 'Format name of data that follows last'
                             'NameValueData field'
  Flags          As Long     'Flags'
  NameValueCCSID As Long     'Character set identifier of NameValueData'
End Type
```

MQRMH - Reference message header:

The following table summarizes the fields in the structure.

Table 188. Fields in MQRMH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Total length of MQRMH, including strings at end of fixed fields, but not the bulk data	StrucLength
Encoding	Numeric encoding of bulk data	Encoding
CodedCharSetId	Character set identifier of bulk data	CodedCharSetId
Format	Format name of bulk data	Format
Flags	Reference message flags	Flags
ObjectType	Object type	ObjectType
ObjectInstanceId	Object instance identifier	ObjectInstanceId
SrcEnvLength	Length of source environment data	SrcEnvLength
SrcEnvOffset	Offset of source environment data	SrcEnvOffset
SrcNameLength	Length of source object name	SrcNameLength
SrcNameOffset	Offset of source object name	SrcNameOffset
DestEnvLength	Length of destination environment data	DestEnvLength
DestEnvOffset	Offset of destination environment data	DestEnvOffset
DestNameLength	Length of destination object name	DestNameLength
DestNameOffset	Offset of destination object name	DestNameOffset
DataLogicalLength	Length of bulk data	DataLogicalLength
DataLogicalOffset	Low offset of bulk data	DataLogicalOffset
DataLogicalOffset2	High offset of bulk data	DataLogicalOffset2

Overview for MQRMH:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: The MQRMH structure defines the format of a reference message header. This header is used with user-written message channel exits to send extremely large amounts of data (called *bulk data*) from one queue manager to another. The difference compared to normal messaging is that the bulk data is not stored on a queue; instead, only a *reference* to the bulk data is stored on the queue. This reduces the possibility of MQ resources being exhausted by a small number of extremely large messages.

Format name: MQFMT_REF_MSG_HEADER.

Character set and encoding: Character data in MQRMH, and the strings addressed by the offset fields, must be in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQRMH must be in the native machine encoding; this is given by the value of MQENC_NATIVE for the C programming language.

Set the character set and encoding of the MQRMH into the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQRMH structure is at the start of the message data), or

- The header structure that precedes the MQRMH structure (all other cases).

Usage: An application puts a message consisting of an MQRMH, but omitting the bulk data. When a message channel agent (MCA) reads the message from the transmission queue, a user-supplied message exit is invoked to process the reference message header. The exit can append to the reference message the bulk data identified by the MQRMH structure, before the MCA sends the message through the channel to the next queue manager.

At the receiving end, a message exit that waits for reference messages must exist. When a reference message is received, the exit must create the object from the bulk data that follows the MQRMH in the message, and then pass on the reference message without the bulk data. The reference message can later be retrieved by an application reading the reference message (without the bulk data) from a queue.

Normally, the MQRMH structure is all that is in the message. However, if the message is on a transmission queue, one or more additional headers precede the MQRMH structure.

A reference message can also be sent to a distribution list. In this case, the MQDH structure and its related records precede the MQRMH structure when the message is on a transmission queue.

Note: Do not send a reference message as a segmented message, because the message exit cannot process it correctly.

Data conversion: For data conversion purposes, converting the MQRMH structure includes conversion of the source environment data, source object name, destination environment data, and destination object name. Any other bytes within *StrucLength* bytes of the start of the structure are either discarded or have undefined values after data conversion. The bulk data is converted provided that all the following are true:

- The bulk data is present in the message when the data conversion is performed.
- The *Format* field in MQRMH has a value other than MQFMT_NONE.
- A user-written data-conversion exit exists with the format name specified.

Be aware, however, that usually the bulk data is *not* present in the message when the message is on a queue, and that as a result the bulk data is converted by the MQGMO_CONVERT option.

Fields for MQRMH:

The MQRMH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the bulk data; it does not apply to character data in the MQRMH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The following special value can be used:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

Do not use MQCCSI_INHERIT if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

This value is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

The initial value of this field is MQCCSI_UNDEFINED.

DataLogicalLength (MQLONG):

The *DataLogicalLength* field specifies the length of the bulk data referenced by the MQRMH structure.

If the bulk data is actually present in the message, the data begins at an offset of *StrucLength* bytes from the start of the MQRMH structure. The length of the entire message minus *StrucLength* gives the length of the bulk data present.

If data is present in the message, *DataLogicalLength* specifies the amount of that data that is relevant. The normal case is for *DataLogicalLength* to have the same value as the length of data present in the message.

If the MQRMH structure represents the remaining data in the object (starting from the specified logical offset), you can use the value zero for *DataLogicalLength*, provided that the bulk data is not actually present in the message.

If no data is present, the end of MQRMH coincides with the end of the message.

The initial value of this field is 0.

DataLogicalOffset (MQLONG):

This field specifies the low offset of the bulk data from the start of the object of which the bulk data forms part. The offset of the bulk data from the start of the object is called the *logical offset*. This is *not* the physical offset of the bulk data from the start of the MQRMH structure; that offset is given by *StrucLength*.

To allow large objects to be sent using reference messages, the logical offset is divided into two fields, and the actual logical offset is given by the sum of these two fields:

- *DataLogicalOffset* represents the remainder obtained when the logical offset is divided by 1 000 000 000. It is thus a value in the range 0 through 999 999 999.
- *DataLogicalOffset2* represents the result obtained when the logical offset is divided by 1 000 000 000. It is thus the number of complete multiples of 1 000 000 000 that exist in the logical offset. The number of multiples is in the range 0 through 999 999 999.

The initial value of this field is 0.

DataLogicalOffset2 (MQLONG):

This field specifies the high offset of the bulk data from the start of the object of which the bulk data forms part. It is a value in the range 0 through 999 999 999. See *DataLogicalOffset* for details.

The initial value of this field is 0.

DestEnvLength (MQLONG):

This is the length of the destination environment data. If this field is zero, there is no destination environment data, and *DestEnvOffset* is ignored.

DestEnvOffset (MQLONG):

This field specifies the offset of the destination environment data from the start of the MQRMH structure. Destination environment data can be specified by the creator of the reference message, if that data is known to the creator. For example, on Windows the destination environment data might be the directory path of the object where the bulk data is to be stored. However, if the creator does not know the destination environment data, it is the responsibility of the user-supplied message exit to determine any environment information needed.

The length of the destination environment data is given by *DestEnvLength*; if this length is zero, there is no destination environment data, and *DestEnvOffset* is ignored. If present, the destination environment data must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the destination environment data is contiguous with any of the data addressed by the *SrcEnvOffset*, *SrcNameOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

DestNameLength (MQLONG):

The length of the destination object name. If this field is zero, there is no destination object name, and *DestNameOffset* is ignored.

DestNameOffset (MQLONG):

This field specifies the offset of the destination object name from the start of the MQRMH structure. The destination object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the destination object name, it is the responsibility of the user-supplied message exit to identify the object to be created or modified.

The length of the destination object name is given by *DestNameLength*; if this length is zero, there is no destination object name, and *DestNameOffset* is ignored. If present, the destination object name must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the destination object name is contiguous with any of the data addressed by the *SrcEnvOffset*, *SrcNameOffset*, and *DestEnvOffset* fields.

The initial value of this field is 0.

Encoding (MQLONG):

This specifies the numeric encoding of the bulk data; it does not apply to numeric data in the MQRMH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is MQENC_NATIVE.

Flags (MQLONG):

These are reference message flags. The following flags are defined:

MQRMHF_LAST
This flag indicates that the reference message represents or contains the last part of the referenced object.

MQRMHF_NOT_LAST
Reference message does not contain or represent last part of object. MQRMHF_NOT_LAST aids program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

The initial value of this field is MQRMHF_NOT_LAST.

Format (MQCHAR8):

This specifies the format name of the bulk data.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The initial value of this field is MQFMT_NONE.

ObjectInstanceId (MQBYTE24):

Use this field to identify a specific instance of an object. If it is not needed, set it to the following value:

MQOII_NONE
No object instance identifier specified. The value is binary zero for the length of the field.

For the C programming language, the constant MQOII_NONE_ARRAY is also defined; this has the same value as MQOII_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_OBJECT_INSTANCE_ID_LENGTH. The initial value of this field is MQOII_NONE.

ObjectType (MQCHAR8):

This is a name that the message exit can use to recognize types of reference message that it supports. The name must conform to the same rules as the *Format* field described above.

The initial value of this field is 8 blanks.

SrcEnvLength (MQLONG):

The length of the source environment data. If this field is zero, there is no source environment data, and *SrcEnvOffset* is ignored.

The initial value of this field is 0.

SrcEnvOffset (MQLONG):

This field specifies the offset of the source environment data from the start of the MQRMH structure. Source environment data can be specified by the creator of the reference message, if that data is known to the creator. For example, on Windows the source environment data might be the directory path of the object containing the bulk data. However, if the creator does not know the source environment data, the user-supplied message exit must determine any environment information needed.

The length of the source environment data is given by *SrcEnvLength*; if this length is zero, there is no source environment data, and *SrcEnvOffset* is ignored. If present, the source environment data must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the environment data starts immediately after the last fixed field in the structure or that it is contiguous with any of the data addressed by the *SrcNameOffset*, *DestEnvOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

SrcNameLength (MQLONG):

The length of the source object name. If this field is zero, there is no source object name, and *SrcNameOffset* is ignored.

The initial value of this field is 0.

SrcNameOffset (MQLONG):

This field specifies the offset of the source object name from the start of the MQRMH structure. The source object name can be specified by the creator of the reference message, if that data is known to the creator. However, if the creator does not know the source object name, the user-supplied message exit must identify the object to be accessed.

The length of the source object name is given by *SrcNameLength*; if this length is zero, there is no source object name, and *SrcNameOffset* is ignored. If present, the source object name must reside completely within *StrucLength* bytes from the start of the structure.

Applications must not assume that the source object name is contiguous with any of the data addressed by the *SrcEnvOffset*, *DestEnvOffset*, and *DestNameOffset* fields.

The initial value of this field is 0.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQRMH_STRUC_ID
Identifier for reference message header structure.

For the C programming language, the constant MQRMH_STRUC_ID_ARRAY is also defined; this has the same value as MQRMH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQRMH_STRUC_ID.

StrucLength (MQLONG):

The total length of MQRMH, including strings at the end of fixed fields, but not the bulk data.

The initial value of this field is zero.

Version (MQLONG):

The structure version number. The value must be:

MQRMH_VERSION_1
Version-1 reference message header structure.

The following constant specifies the version number of the current version:

MQRMH_CURRENT_VERSION
Current version of reference message header structure.

The initial value of this field is MQRMH_VERSION_1.

Initial values and language declarations for MQRMH:

Table 189. Initial values of fields in MQRMH for MQRMH

Field name	Name of constant	Value of constant
StrucId	MQRMH_STRUC_ID	'RMH¬'
Version	MQRMH_VERSION_1	1
StrucLength	None	0
Encoding	MQENC_NATIVE	Depends on environment
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQRMHF_NOT_LAST	0
ObjectType	None	Blanks
ObjectInstanceId	MQOII_NONE	Nulls
SrcEnvLength	None	0
SrcEnvOffset	None	0
SrcNameLength	None	0
SrcNameOffset	None	0
DestEnvLength	None	0
DestEnvOffset	None	0
DestNameLength	None	0
DestNameOffset	None	0
DataLogicalLength	None	0
DataLogicalOffset	None	0
DataLogicalOffset2	None	0

Table 189. Initial values of fields in MQRMH for MQRMH (continued)

Field name	Name of constant	Value of constant
Notes:		
1. The symbol ¬ represents a single blank character.		
2. In the C programming language, the macro variable MQRMH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure: `MQRMH MyRMH = {MQRMH_DEFAULT};`		

C declaration:

```
typedef struct tagMQRMH MQRMH;
struct tagMQRMH {
  MQCHAR4   StrucId;             /* Structure identifier */
  MQLONG    Version;             /* Structure version number */
  MQLONG    StrucLength;         /* Total length of MQRMH, including
                                    strings at end of fixed fields, but
                                    not the bulk data */
  MQLONG    Encoding;            /* Numeric encoding of bulk data */
  MQLONG    CodedCharSetId;      /* Character set identifier of bulk
                                    data */
  MQCHAR8   Format;              /* Format name of bulk data */
  MQLONG    Flags;               /* Reference message flags */
  MQCHAR8   ObjectType;          /* Object type */
  MQBYTE24  ObjectInstanceId;    /* Object instance identifier */
  MQLONG    SrcEnvLength;        /* Length of source environment data */
  MQLONG    SrcEnvOffset;        /* Offset of source environment data */
  MQLONG    SrcNameLength;       /* Length of source object name */
  MQLONG    SrcNameOffset;       /* Offset of source object name */
  MQLONG    DestEnvLength;       /* Length of destination environment
                                    data */
  MQLONG    DestEnvOffset;       /* Offset of destination environment
                                    data */
  MQLONG    DestNameLength;      /* Length of destination object name */
  MQLONG    DestNameOffset;      /* Offset of destination object name */
  MQLONG    DataLogicalLength;   /* Length of bulk data */
  MQLONG    DataLogicalOffset;   /* Low offset of bulk data */
  MQLONG    DataLogicalOffset2;  /* High offset of bulk data */
};
```

COBOL declaration:

```
**   MQRMH structure
  10 MQRMH.
**    Structure identifier
   15 MQRMH-STRUCID            PIC X(4).
**    Structure version number
   15 MQRMH-VERSION            PIC S9(9) BINARY.
**    Total length of MQRMH, including strings at end of fixed fields,
**    but not the bulk data
   15 MQRMH-STRUCLENGTH        PIC S9(9) BINARY.
**    Numeric encoding of bulk data
   15 MQRMH-ENCODING           PIC S9(9) BINARY.
**    Character set identifier of bulk data
   15 MQRMH-CODEDCHARSETID     PIC S9(9) BINARY.
**    Format name of bulk data
   15 MQRMH-FORMAT             PIC X(8).
**    Reference message flags
   15 MQRMH-FLAGS              PIC S9(9) BINARY.
**    Object type
   15 MQRMH-OBJECTTYPE         PIC X(8).
**    Object instance identifier
   15 MQRMH-OBJECTINSTANCEID   PIC X(24).
```

```
**    Length of source environment data
   15 MQRMH-SRCENVLENGTH       PIC S9(9) BINARY.
**    Offset of source environment data
   15 MQRMH-SRCENVOFFSET       PIC S9(9) BINARY.
**    Length of source object name
   15 MQRMH-SRCNAMELENGTH      PIC S9(9) BINARY.
**    Offset of source object name
   15 MQRMH-SRCNAMEOFFSET      PIC S9(9) BINARY.
**    Length of destination environment data
   15 MQRMH-DESTENVLENGTH      PIC S9(9) BINARY.
**    Offset of destination environment data
   15 MQRMH-DESTENVOFFSET      PIC S9(9) BINARY.
**    Length of destination object name
   15 MQRMH-DESTNAMELENGTH     PIC S9(9) BINARY.
**    Offset of destination object name
   15 MQRMH-DESTNAMEOFFSET     PIC S9(9) BINARY.
**    Length of bulk data
   15 MQRMH-DATALOGICALLENGTH  PIC S9(9) BINARY.
**    Low offset of bulk data
   15 MQRMH-DATALOGICALOFFSET  PIC S9(9) BINARY.
**    High offset of bulk data
   15 MQRMH-DATALOGICALOFFSET2 PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQRMH based,
  3 StrucId            char(4),       /* Structure identifier */
  3 Version            fixed bin(31), /* Structure version number */
  3 StrucLength        fixed bin(31), /* Total length of MQRMH,
                                         including strings at end of
                                         fixed fields, but not the bulk
                                         data */
  3 Encoding           fixed bin(31), /* Numeric encoding of bulk
                                         data */
  3 CodedCharSetId     fixed bin(31), /* Character set identifier of
                                         bulk data */
  3 Format             char(8),       /* Format name of bulk data */
  3 Flags              fixed bin(31), /* Reference message flags */
  3 ObjectType         char(8),       /* Object type */
  3 ObjectInstanceId   char(24),      /* Object instance identifier */
  3 SrcEnvLength       fixed bin(31), /* Length of source environment
                                         data */
  3 SrcEnvOffset       fixed bin(31), /* Offset of source environment
                                         data */
  3 SrcNameLength      fixed bin(31), /* Length of source object name */
  3 SrcNameOffset      fixed bin(31), /* Offset of source object name */
  3 DestEnvLength      fixed bin(31), /* Length of destination
                                         environment data */
  3 DestEnvOffset      fixed bin(31), /* Offset of destination
                                         environment data */
  3 DestNameLength     fixed bin(31), /* Length of destination object
                                         name */
  3 DestNameOffset     fixed bin(31), /* Offset of destination object
                                         name */
  3 DataLogicalLength  fixed bin(31), /* Length of bulk data */
  3 DataLogicalOffset  fixed bin(31), /* Low offset of bulk data */
  3 DataLogicalOffset2 fixed bin(31); /* High offset of bulk data */
```

High Level Assembler declaration:

```
MQRMH                   DSECT
MQRMH_STRUCID           DS   CL4   Structure identifier
MQRMH_VERSION           DS   F     Structure version number
MQRMH_STRUCLENGTH       DS   F     Total length of MQRMH, including
*                                  strings at end of fixed fields, but
*                                  not the bulk data
MQRMH_ENCODING          DS   F     Numeric encoding of bulk data
MQRMH_CODEDCHARSETID    DS   F     Character set identifier of bulk
*                                  data
MQRMH_FORMAT            DS   CL8   Format name of bulk data
MQRMH_FLAGS             DS   F     Reference message flags
MQRMH_OBJECTTYPE        DS   CL8   Object type
MQRMH_OBJECTINSTANCEID  DS   XL24  Object instance identifier
MQRMH_SRCENVLENGTH      DS   F     Length of source environment data
MQRMH_SRCENVOFFSET      DS   F     Offset of source environment data
MQRMH_SRCNAMELENGTH     DS   F     Length of source object name
MQRMH_SRCNAMEOFFSET     DS   F     Offset of source object name
MQRMH_DESTENVLENGTH     DS   F     Length of destination environment
*                                  data
MQRMH_DESTENVOFFSET     DS   F     Offset of destination environment
*                                  data
MQRMH_DESTNAMELENGTH    DS   F     Length of destination object name
MQRMH_DESTNAMEOFFSET    DS   F     Offset of destination object name
MQRMH_DATALOGICALLENGTH DS   F     Length of bulk data
MQRMH_DATALOGICALOFFSET DS   F     Low offset of bulk data
MQRMH_DATALOGICALOFFSET2 DS  F     High offset of bulk data
*
MQRMH_LENGTH            EQU  *-MQRMH
                        ORG  MQRMH
MQRMH_AREA              DS   CL(MQRMH_LENGTH)
```

Visual Basic declaration:

```
Type MQRMH
  StrucId            As String*4 'Structure identifier'
  Version            As Long     'Structure version number'
  StrucLength        As Long     'Total length of MQRMH, including'
                                 'strings at end of fixed fields, but'
                                 'not the bulk data'
  Encoding           As Long     'Numeric encoding of bulk data'
  CodedCharSetId     As Long     'Character set identifier of bulk data'
  Format             As String*8 'Format name of bulk data'
  Flags              As Long     'Reference message flags'
  ObjectType         As String*8 'Object type'
  ObjectInstanceId   As MQBYTE24 'Object instance identifier'
  SrcEnvLength       As Long     'Length of source environment data'
  SrcEnvOffset       As Long     'Offset of source environment data'
  SrcNameLength      As Long     'Length of source object name'
  SrcNameOffset      As Long     'Offset of source object name'
  DestEnvLength      As Long     'Length of destination environment'
                                 'data'
  DestEnvOffset      As Long     'Offset of destination environment'
                                 'data'
  DestNameLength     As Long     'Length of destination object name'
  DestNameOffset     As Long     'Offset of destination object name'
  DataLogicalLength  As Long     'Length of bulk data'
  DataLogicalOffset  As Long     'Low offset of bulk data'
  DataLogicalOffset2 As Long     'High offset of bulk data'
End Type
```

MQRR - Response record:

The following table summarizes the fields in the structure.

Table 190. Fields in MQRR

Field	Description	Topic
CompCode	Completion code for queue	CompCode
Reason	Reason code for queue	Reason

Overview for MQRR:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

Purpose: Use the MQRR structure to receive the completion code and reason code resulting from the open or put operation for a single destination queue, when the destination is a distribution list. MQRR is an output structure for the MQOPEN, MQPUT, and MQPUT1 calls.

Character set and encoding: Data in MQRR must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Usage: By providing an array of these structures on the MQOPEN and MQPUT calls, or on the MQPUT1 call, you can determine the completion codes and reason codes for all the queues in a distribution list when the outcome of the call is mixed, that is, when the call succeeds for some queues in the list but fails for others. Reason code MQRC_MULTIPLE_REASONS from the call indicates that the response records (if provided by the application) have been set by the queue manager.

Fields for MQRR:

The MQRR structure contains the following fields; the fields are described in **alphabetical order**:

CompCode (MQLONG):

This is the completion code resulting from the open or put operation for the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is MQCC_OK.

Reason (MQLONG):

This is the reason code resulting from the open or put operation for the queue with the name that was specified by the corresponding element in the array of MQOR structures provided on the MQOPEN or MQPUT1 call.

This is always an output field. The initial value of this field is MQRC_NONE.

Initial values and language declarations for MQRR:

Table 191. Initial values of fields in MQRR for MQRR

Field name	Name of constant	Value of constant
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0

Notes:

1. In the C programming language, the macro variable MQRR_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQRR MyRR = {MQRR_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQRR MQRR;
struct tagMQRR {
  MQLONG  CompCode;  /* Completion code for queue */
  MQLONG  Reason;    /* Reason code for queue */
};
```

COBOL declaration:

```
**   MQRR structure
  10 MQRR.
**    Completion code for queue
   15 MQRR-COMPCODE PIC S9(9) BINARY.
**    Reason code for queue
   15 MQRR-REASON   PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQRR based,
  3 CompCode fixed bin(31), /* Completion code for queue */
  3 Reason   fixed bin(31); /* Reason code for queue */
```

Visual Basic declaration:

```
Type MQRR
  CompCode As Long 'Completion code for queue'
  Reason   As Long 'Reason code for queue'
End Type
```

MQSCO - SSL configuration options:

The following table summarizes the fields in the structure.

Table 192. Fields in MQSCO.

List of fields in MQSCO, by version, with links to the topics that describe the fields.

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
KeyRepository	Location of key repository	KeyRepository
CryptoHardware	Details of cryptographic hardware	CryptoHardware
AuthInfoRecCount	Number of MQAIR records present	AuthInfoRecCount
AuthInfoRecOffset	Offset of first MQAIR record from start of MQSCO	AuthInfoRecOffset

Table 192. Fields in MQSCO (continued).

List of fields in MQSCO, by version, with links to the topics that describe the fields.

Field	Description	Topic
AuthInfoRecPtr	Address of first MQAIR record	AuthInfoRecPtr
Note: The following two fields are ignored if *Version* is less than MQSCO_VERSION_2.		
KeyResetCount	SSL secret key reset count	KeyResetCount
FipsRequired	Use FIPS-certified cryptographic algorithms in WebSphere MQ	"FipsRequired (MQLONG)" on page 1855
Note: The following field is ignored if *Version* is less than MQSCO_VERSION_3.		
EncryptionPolicySuiteB	Use only Suite B cryptographic algorithms	EncryptionPolicySuiteB
Note: The following field is ignored if *Version* is less than MQSCO_VERSION_4.		
CertificateValPolicy	Certificate validation policy	CertificateValPolicy

Related reference:

"MQCNO - Connect options" on page 1607
The following table summarizes the fields in the structure.

"Overview for MQSCO"
Availability: AIX, HP-UX, IBM i, Solaris, Linux and Windows clients.

"Fields for MQSCO"

"Initial values and language declarations for MQSCO" on page 1857

Overview for MQSCO:

Availability: AIX, HP-UX, IBM i, Solaris, Linux and Windows clients.

Purpose: The MQSCO structure (in conjunction with the SSL fields in the MQCD structure) allows an application running as a WebSphere MQ MQI client to specify configuration options that control the use of SSL for the client connection when the channel protocol is TCP/IP. The structure is an input parameter on the MQCONNX call.

If the channel protocol for the client channel is not TCP/IP, the MQSCO structure is ignored.

Character set and encoding: Data in MQSCO mustis sbe in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE.

Fields for MQSCO:

The MQSCO structure contains the following fields; the fields are described in **alphabetical order**:

AuthInfoRecCount (MQLONG):

This is the number of authentication information (MQAIR) records addressed by the *AuthInfoRecPtr* or *AuthInfoRecOffset* fields. For more information, see "MQAIR - Authentication information record" on page 1557. The value must be zero or greater. If the value is not valid, the call fails with reason code MQRC_AUTH_INFO_REC_COUNT_ERROR.

This is an input field. The initial value of this field is 0.

AuthInfoRecOffset (MQLONG):

This is the offset in bytes of the first authentication information record from the start of the MQSCO structure. The offset can be positive or negative. The field is ignored if *AuthInfoRecCount* is zero.

You can use either *AuthInfoRecOffset* or *AuthInfoRecPtr* to specify the MQAIR records, but not both; see the description of the *AuthInfoRecPtr* field for details.

This is an input field. The initial value of this field is 0.

AuthInfoRecPtr (PMQAIR):

This is the address of the first authentication information record. The field is ignored if *AuthInfoRecCount* is zero.

You can provide the array of MQAIR records in one of two ways:

- By using the pointer field *AuthInfoRecPtr*

  In this case, the application can declare an array of MQAIR records that is separate from the MQSCO structure, and set *AuthInfoRecPtr* to the address of the array.

  Consider using *AuthInfoRecPtr* for programming languages that support the pointer data type in a fashion that is portable to different environments (for example, the C programming language).
- By using the offset field *AuthInfoRecOffset*

  In this case, the application must declare a compound structure containing an MQSCO followed by the array of MQAIR records, and set *AuthInfoRecOffset* to the offset of the first record in the array from the start of the MQSCO structure. Ensure that this value is correct, and has a value that can be accommodated within an MQLONG (the most restrictive programming language is COBOL, for which the valid range is -999 999 999 through +999 999 999).

  Consider using *AuthInfoRecOffset* for programming languages that do not support the pointer data type, or that implement the pointer data type in a fashion that is not portable to different environments (for example, the COBOL programming language).

Whatever technique you choose, only one of *AuthInfoRecPtr* and *AuthInfoRecOffset* can be used; the call fails with reason code MQRC_AUTH_INFO_REC_ERROR if both are nonzero.

This is an input field. The initial value of this field is the null pointer in those programming languages that support pointers, and an all-null byte string otherwise.

Note: On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

CertificateValPolicy (MQLONG):

This field specifies what type of certificate validation policy is used. The field can be set to one of the following values:

MQ_CERT_VAL_POLICY_ANY
: Apply each of the certificate validation policies supported by the secure sockets library. Accept the certificate chain if any of the policies considers the certificate chain valid.

MQ_CERT_VAL_POLICY_RFC5280
: Apply only the RFC5280 compliant certificate validation policy. This setting provides stricter validation than the ANY setting, but rejects some older digital certificates.

The initial value of this field is MQ_CERT_VAL_POLICY_ANY

CryptoHardware (MQCHAR256):

This field gives configuration details for cryptographic hardware connected to the client system.

Set the field to a string of the following format, or leave it blank or null:

```
GSK_PKCS11=<the PKCS #11 driver path and file name>;<the PKCS #11
token label>;<the PKCS #11 token password>;<symmetric cipher setting>;
```

To use cryptographic hardware which conforms to the PKCS #11 interface, for example, the IBM 4960 or IBM 4764, the PKCS #11 driver path, PKCS #11 token label, and PKCS #11 token password strings must be specified, each terminated by a semi-colon.

The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver file name is the name of the shared library. An example of the value required for the PKCS #11 path and file name is:

```
/usr/lib/pkcs11/PKCS11_API.so
```

The PKCS #11 token label must be entirely in lowercase. If you have configured your hardware with a mixed case or uppercase token label, re-configure it with this lowercase label.

If no cryptographic hardware configuration is required, set the field to blank or null.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. If the value is not valid, or leads to a failure when used to configure the cryptographic hardware, the call fails with reason code MQRC_CRYPTO_HARDWARE_ERROR.

This is an input field. The length of this field is given by MQ_SSL_CRYPTO_HARDWARE_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

EncryptionPolicySuiteB(MQLONG):

This field Specifies whether Suite B compliant cryptography is used and what level of strength is employed. The value can be one or more of:

- MQ_SUITE_B_NONE
  Suite B compliant cryptography is not used.
- MQ_SUITE_B_128_BIT
  Suite B 128-bit strength security is used.
- MQ_SUITE_B_192_BIT
  Suite B 192-bit strength security is used.

Note: Using the MQ_SUITE_B_NONE with any other value in this field is invalid.

FipsRequired (MQLONG):

WebSphere MQ can be configured with cryptographic hardware so that the cryptography modules used are those provided by the hardware product; these can be FIPS-certified to a particular level depending on the cryptographic hardware product in use. Use this field to specify that only FIPS-certified algorithms are used if the cryptography is provided in WebSphere MQ-provided software.

When WebSphere MQ is installed an implementation of SSL cryptography is also installed which provides some FIPS-certified modules.

The values can be:

MQSSL_FIPS_NO

This is the default value. When set to this value:

- Any CipherSpec supported on a particular platform can be used.
- If run without use of cryptographic hardware, the following CipherSpecs run using FIPS 140-2 certified cryptography on the WebSphere MQ platforms:
  - TLS_RSA_WITH_3DES_EDE_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
  - TLS_RSA_WITH_AES_256_CBC_SHA

MQSSL_FIPS_YES

When set to this value, unless you are using cryptographic hardware to perform the cryptography, you can be sure that

- Only FIPS-certified cryptographic algorithms can be used in the CipherSpec applying to this client connection.
- Inbound and outbound SSL channel connections only succeed if one of the following Cipher Specs are used:
  - TLS_RSA_WITH_3DES_EDE_CBC_SHA
  - TLS_RSA_WITH_AES_128_CBC_SHA
  - TLS_RSA_WITH_AES_256_CBC_SHA

Notes:

1. CipherSpec TLS_RSA_WITH_3DES_EDE_CBC_SHA is deprecated.
2. Where possible, if FIPS-only CipherSpecs are configured then the MQI client rejects connections which specify a non-FIPS CipherSpec with MQRC_SSL_INITIALIZATION_ERROR. WebSphere MQ does not guarantee to reject all such connections and it is your responsibility to determine whether your WebSphere MQ configuration is FIPS-compliant.

KeyRepository (MQCHAR256):

This field is relevant only for WebSphere MQ MQI clients running on UNIX systems and Windows systems. It specifies the location of the key database file in which keys and certificates are stored. The key database file must have a file name of the form `zzz.kdb`, where `zzz` is user-selectable. The *KeyRepository* field contains the path to this file, along with the file name stem (all characters in the file name up to but not including the final `.kdb`). The `.kdb` file suffix is added automatically.

Each key database file has an associated *password stash file*. This holds encoded passwords that are used to allow programmatic access to the key database. The password stash file must reside in the same directory and have the same file stem as the key database, and must end with the suffix `.sth`.

For example, if the *KeyRepository* field has the value `/xxx/yyy/key`, the key database file must be `/xxx/yyy/key.kdb`, and the password stash file must be `/xxx/yyy/key.sth`, where `xxx` and `yyy` represent directory names.

If the value is shorter than the length of the field, terminate the value with a null character, or pad it with blanks to the length of the field. The value is not checked; if there is an error in accessing the key repository, the call fails with reason code MQRC_KEY_REPOSITORY_ERROR.

To run an SSL connection from a WebSphere MQ MQI client, set *KeyRepository* to a valid key database file name.

This is an input field. The length of this field is given by MQ_SSL_KEY_REPOSITORY_LENGTH. The initial value of this field is the null string in C, and blank characters in other programming languages.

KeyResetCount (MQLONG):

This represents the total number of unencrypted bytes sent and received within an SSL or TLS conversation before the secret key is renegotiated.

The number of bytes includes control information sent by the MCA.

If you specify an SSL or TLS secret key reset count in the range 1 byte through 32 KB, SSL or TLS channels will use a secret key reset count of 32 KB. This is to avoid the processing cost of excessive key resets which would occur for small SSL or TLS secret key reset values.

This is an input field. The value is a number in the range 0 through 999 999 999, with a default value of 0. Use a value of 0 to indicate that secret keys are never renegotiated.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQSCO_STRUC_ID
Identifier for SSL configuration options structure.

For the C programming language, the constant MQSCO_STRUC_ID_ARRAY is also defined; this has the same value as MQSCO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSCO_STRUC_ID.

Version (MQLONG):

This is the structure version number; the value must be:

MQSCO_VERSION_1
Version-1 SSL configuration options structure.

MQSCO_VERSION_2
Version-2 SSL configuration options structure.

MQSCO_VERSION_3
Version-3 SSL configuration options structure.

MQSCO_VERSION_4
Version-4 SSL configuration options structure.

The following constant specifies the version number of the current version:

MQSCO_CURRENT_VERSION
Current version of SSL configuration options structure.

This is always an input field. The initial value of this field is MQSCO_VERSION_1.

Initial values and language declarations for MQSCO:

Table 193. Initial values of fields in MQSCO.

Description of fields in MQSCO and their initial values

Field name	Name of constant	Value of constant
StrucId	MQSCO_STRUC_ID	'SCO¬'
Version	MQSCO_CURRENT_VERSION	1
KeyRepository	None	Null string or blanks
CryptoHardware	None	Null string or blanks
AuthInfoRecCount	None	0
AuthInfoRecOffset	None	0
AuthInfoRecPtr	None	Null pointer or null bytes
KeyResetCount	MQSCO_RESET_COUNT_DEFAULT	0
FipsRequired	MQSSL_FIPS_NO	0
EncryptionPolicySuiteB	MQ_SUITE_B_NONE, MQ_SUITE_B_NOT_AVAILABLE, MQ_SUITE_B_NOT_AVAILABLE, MQ_SUITE_B_NOT_AVAILABLE	1, 0, 0, 0
CertificateValPolicy	MQ_CERT_VAL_POLICY_DEFAULT	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQSCO_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

   ```
   MQSCO MySCO = {MQSCO_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQSCO MQSCO;
struct tagMQSCO {
   MQCHAR4    StrucId;                  /* Structure identifier */
   MQLONG     Version;                  /* Structure version number */
   MQCHAR256  KeyRepository;            /* Location of SSL key */
                                        /* repository */
   MQCHAR256  CryptoHardware;           /* Cryptographic hardware */
                                        /* configuration string */
   MQLONG     AuthInfoRecCount;         /* Number of MQAIR records */
                                        /* present */
   MQLONG     AuthInfoRecOffset;        /* Offset of first MQAIR */
                                        /* record from start of */
                                        /* MQSCO structure */
   PMQAIR     AuthInfoRecPtr;           /* Address of first MQAIR */
                                        /* record */
/* Ver:1 */
   MQLONG     KeyResetCount;            /* Number of unencrypted */
                                        /* bytes sent/received */
                                        /* before secret key is */
                                        /* reset */
   MQLONG     FipsRequired;             /* Using FIPS-certified */
/* Ver:2 */
                                        /* algorithms */
   MQLONG     EncryptionPolicySuiteB[4]; /* Use only Suite B */
/* Ver:3 */
                                        /* cryptographic algorithms */
   MQLONG     CertificateValPolicy;     /* Certificate validation */
                                        /* policy */
/* Ver:4 */
```

COBOL declaration:

```
**   MQSCO structure
  10 MQSCO.
**    Structure identifier
   15 MQSCO-STRUCID                  PIC X(4).
**    Structure version number
   15 MQSCO-VERSION                  PIC S9(9) BINARY.
**    Location of SSL key repository
   15 MQSCO-KEYREPOSITORY            PIC X(256).
**    Cryptographic hardware configuration string
   15 MQSCO-CRYPTOHARDWARE           PIC X(256).
**    Number of MQAIR records present
   15 MQSCO-AUTHINFORECCOUNT         PIC S9(9) BINARY.
**    Offset of first MQAIR record from start of MQSCO structure
   15 MQSCO-AUTHINFORECOFFSET        PIC S9(9) BINARY.
** Address of first MQAIR record
    15  MQSCO-AUTHINFORECPTR POINTER.
** Version 1 **
** Number of unencrypted bytes sent/received before secret key is
** reset
15  MQSCO-KEYRESETCOUNT PIC S9(9) BINARY.
** Using FIPS-certified algorithms
15  MQSCO-FIPSREQUIRED PIC S9(9) BINARY.
** Version 2 **
** Use only Suite B cryptographic algorithms
15  MQSCO-ENCRYPTIONPOLICYSUITEB PIC S9(9) BINARY OCCURS 4.
** Version 3 **
** Certificate validation policy setting
15  MQSCO-CERTIFICATEVALPOLICY PIC S9(9) BINARY.
** Version 4
```

PL/I declaration:

```
dcl
 1 MQSCO based,
  3 StrucId                    char(4),       /* Structure identifier */
  3 Version                    fixed bin(31), /* Structure version number */
  3 KeyRepository              char(256),     /* Location of SSL key
                                                 repository */
  3 CryptoHardware             char(256),     /* Cryptographic hardware
                                                 configuration string */
  3 AuthInfoRecCount           fixed bin(31), /* Number of MQAIR records
                                                 present */
  3 AuthInfoRecOffset          fixed bin(31), /* Offset of first MQAIR record
                                                 from start of MQSCO structure */
  3 AuthInfoRecPtr             pointer,       /* Address of first MQAIR record */
  3 KeyResetCount              fixed bin(31), /* Key reset count */
/* Version 1 */
  3 FipsRequired               fixed bin(31), /* FIPS required */
/* Version 2 */
  3 EncryptionPolicySuiteB (4) fixed bin(31), /* Suite B encryption policy */
/* Version 3 */
  3 CertificateValPolicy       fixed bin(31); /* Certificate validation policy */
/* Version 4 */
```

Visual Basic declaration:

```
Type MQSCO
  StrucId           As String*4   'Structure identifier'
  Version           As Long       'Structure version number'
  KeyRepository     As String*256 'Location of SSL key repository'
  CryptoHardware    As String*256 'Cryptographic hardware configuration'
                                  'string'
  AuthInfoRecCount  As Long       'Number of MQAIR records present'
  AuthInfoRecOffset As Long       'Offset of first MQAIR record from'
                                  'start of MQSCO structure'
  AuthInfoRecPtr    As MQPTR      'Address of first MQAIR record'
  KeyResetCount     As Long       'Number of unencrypted bytes sent/received before secret key is reset'
'Version 1'
  FipsRequired      As Long       'Mandatory FIPS CipherSpecs?'
'Version 2'
End Type
```

MQSD - Subscription descriptor:

The following table summarizes the fields in the structure.

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options
ObjectName	Object name	ObjectName
AlternateUserId	Alternate User Id	AlternateUserId
AlternateSecurityId	Alternate Security ID	AlternateSecurityId
SubExpiry	Subscription Expiry	SubExpiry
ObjectString	Object String	ObjectString
SubName	Subscription Name	SubName
SubUserData	Subscription user data	SubUserData
SubCorrelId	Subscription Correlation ID	SubCorrelId
PubPriority	Publication priority	PubPriority
PubAccountingToken	Publication Accounting Token	PubAccountingToken
PubAppIdentityData	Publication application identity data	PubAppIdentityData
SelectionString	String providing selection criteria	SelectionString
SubLevel	Subscription Level	SubLevel
ResObjectString	Long object name	ResObjectString

Overview for MQSD:

Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS, plus WebSphere MQ MQI clients connected to these systems.

Purpose: The MQSD structure is used to specify details about the subscription being made.

The structure is an input/output parameter on the MQSUB call. For more information, see MQSUB usage notes.

Managed subscriptions: If an application has no specific need to use a particular queue as the destination for those publications that match its subscription, it can use the managed subscription feature.

If an application elects to use a managed subscription, the queue manager informs the subscriber about the destination where published messages are sent, by providing an object handle as an output from the MQSUB call. For more information, see Hobj (MQHOBJ) - input/output.

When the subscription is removed, the queue manager also undertakes to clean up messages that have not been retrieved from the managed destination, in the following situations:

- When the subscription is removed - by use of MQCLOSE with MQCO_REMOVE_SUB - and the managed Hobj is closed.
- By implicit means when the connection is lost to an application using a non-durable subscription (MQSO_NON_DURABLE)
- By expiration when a subscription is removed because it has expired and the managed Hobj is closed.

You must use managed subscriptions with non-durable subscriptions, so that this clean up can occur, and so that messages for closed non-durable subscriptions do not take up space in your queue manager. Durable subscriptions can also use managed destinations.

Version: The current version of MQSD is MQSD_VERSION_1.

Character set and encoding: Data in MQSD must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQSD:

The MQSD structure contains the following fields; the fields are described in alphabetical order:

AlternateSecurityId (MQBYTE40):

This is a security identifier that is passed with the AlternateUserId to the authorization service to allow appropriate authorization checks to be performed.

AlternateSecurityId is used only if MQSO_ALTERNATE_USER_AUTHORITY is specified, and the AlternateUserId field is not entirely blank up to the first null character or the end of the field.

On return from an MQSUB call using MQSO_RESUME, this field is unchanged.

See the description of “AlternateSecurityId (MQBYTE40)” on page 1770 in the MQOD data type for more information.

AlternateUserId (MQCHAR12):

If you specify MQSO_ALTERNATE_USER_AUTHORITY, this field contains an alternative user identifier that is used to check the authorization for the subscription and for output to the destination queue (specified in the *Hobj* parameter of the MQSUB call), in place of the user identifier that the application is currently running under.

If successful, the user identifier specified in this field is recorded as the subscription owning user identifier in place of the user identifier that the application is currently running under.

If MQSO_ALTERNATE_USER_AUTHORITY is specified and this field is entirely blank up to the first null character or the end of the field, the subscription can succeed only if no user authorization is needed to subscribe to this topic with the options specified or the destination queue for output.

If MQSO_ALTERNATE_USER_AUTHORITY is not specified, this field is ignored.

The following differences exist in the environments indicated:

- On z/OS, only the first 8 characters of AlternateUserId are used to check the authorization for the subscription. However, the current user identifier must be authorized to specify this particular alternative user identifier; all 12 characters of the alternative user identifier are used for this check. The user identifier must contain only characters allowed by the external security manager.

On return from an MQSUB call using MQSO_RESUME, this field is unchanged.

This is an input field. The length of this field is given by MQ_USER_ID_LENGTH. The initial value of this field is the null string in C, and 12 blank characters in other programming languages.

ObjectName (MQCHAR48):

This is the name of the topic object as defined on the local queue manager.

The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. Use a null character to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS:
  - Avoid names that begin or end with an underscore; they cannot be processed by the operations and control panels.
  - The percent character has a special meaning to RACF. If RACF is used as the external security manager, names must not contain the percent. If they do, those names are not included in any security checks when RACF generic profiles are used.
- On IBM i, names containing lowercase characters, forward slash, or percent, must be enclosed in quotation marks when specified on commands. These quotation marks must not be specified for names that occur as fields in structures or as parameters on calls.

The *ObjectName* is used to form the full topic name.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see "Using topic strings" on page 1876.

If the object identified by the *ObjectName* field cannot be found, the call fails with reason code MQRC_UNKNOWN_OBJECT_NAME even if there is a string specified in *ObjectString*.

On return from an MQSUB call using the MQSO_RESUME option this field is unchanged.

The length of this field is given by MQ_TOPIC_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

If altering an existing subscription using the MQSO_ALTER option, the name of the topic object subscribed to cannot be changed. This field and the *ObjectString* field can be omitted. If they are provided, they must resolve to the same full topic name. If they do not, the call fails with MQRC_TOPIC_NOT_ALTERABLE.

ObjectString (MQCHARV):

This is the long object name to be used.

The *ObjectString* is used to form the Full topic name.

The full topic name can be built from two different fields: *ObjectName* and *ObjectString*. For details of how these two fields are used, see "Using topic strings" on page 1876.

The maximum length of *ObjectString* is 10240.

If *ObjectString* is not specified correctly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_OBJECT_STRING_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If there are wildcards in the *ObjectString* the interpretation of those wildcards can be controlled using the Wildcard options specified in the Options field of the MQSD.

On return from an MQSUB call using the MQSO_RESUME option this field is unchanged. The full topic name used is returned in the *ResObjectString* field if a buffer is provided.

If altering an existing subscription using the MQSO_ALTER option, the long name of the topic object subscribed to cannot be changed. This field and the *ObjectName* field can be omitted. If they are provided they must resolve to the same full topic name or the call fails with MQRC_TOPIC_NOT_ALTERABLE.

Options (MQLONG):

This provides options to control the action of the MQSUB call.

You must specify at least one of the following options:

- MQSO_ALTER
- MQSO_RESUME
- MQSO_CREATE

The values you specify for the options can be used in the following ways:

- The values can be added. Do not add the same constant more than once.
- The values can be combined using the bitwise OR operation, if the programming language supports bitwise operations.

Combinations that are not valid are noted in this topic; any other combinations are valid.

Access or creation options: Access and creation options control whether a subscription is created, or whether an existing subscription is returned or altered. You must specify at least one of these options. The table displays valid combinations of access and creation options.

Combination of options	Notes
MQSO_CREATE	Creates a subscription if one does not exist. This combination fails if the subscription exists.
MQSO_RESUME	Resumes an existing subscription. This combination fails if no subscription exists.
MQSO_CREATE + MQSO_RESUME	Creates a subscription if one does not exist and resumes a matching one, if it does exist. This combination is useful when it is used in an application that is run a number of times.
MQSO_ALTER (see note)	Resumes an existing subscription, altering any fields to match that specified in the MQSD. This combination fails if no subscription exists.
MQSO_CREATE + MQSO_ALTER (see note)	Creates a subscription if one does not exist and resumes a matching one, if it does exist, altering any fields to match that specified in the MQSD. This combination is useful combination when used in an application that wants to ensure that its subscription is in a certain state before proceeding.
Note: Options specifying MQSO_ALTER can also specify MQSO_RESUME, but this combination has no additional effect to specifying MQSO_ALTER alone. MQSO_ALTER implies MQSO_RESUME, because calling MQSUB to alter a subscription implies that the subscription will also be resumed. The opposite is not true, however: resuming a subscription does not imply it is to be altered.	

MQSO_CREATE

Create a new subscription for the topic specified. If a subscription using the same *SubName* exists, the call fails with MQRC_SUB_ALREADY_EXISTS. This failure can be avoided by combining the MQSO_CREATE option with MQSO_RESUME. The *SubName* is not always necessary. For more details, see the description of that field.

Combining MQSO_CREATE with MQSO_RESUME returns a handle to a pre-existing subscription for the specified *SubName* if one is found; if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

MQSO_CREATE can also be combined with MQSO_ALTER to similar effect.

MQSO_RESUME

Return a handle to a pre-existing subscription which matches that specified by *SubName*. No changes are made to the matching subscriptions attributes and they are returned on output in the MQSD structure. Only the following MQSD fields are used: StrucId, Version, Options, AlternateUserId and AlternateSecurityId, and SubName.

The call fails with reason code MQRC_NO_SUBSCRIPTION if a subscription does not exist matching the full subscription name. This failure can be avoided by combining the MQSO_CREATE option with MQSO_RESUME.

The user ID of the subscription is the user ID that created the subscription, or if it has been later altered by a different user ID, it is the user ID of the most recent successful alteration. If an AlternateUserId is used, and use of alternate user IDs is allowed for that user, the alternate user ID is recorded as the user ID that created the subscription instead of the user ID under which the subscription was made.

If a matching subscription exists that was created without the MQSO_ANY_USERID option, and the user ID of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code MQRC_IDENTITY_MISMATCH.

If a matching subscription exists and is currently in use, the call fails with MQRC_SUBSCRIPTION_IN_USE.

If the subscription named in SubName is not a valid subscription to resume or alter from an application, the call fails with MQRC_INVALID_SUBSCRIPTION.

MQSO_RESUME is implied by MQSO_ALTER so you do not need to combine it with that option. However, combining the two options does not cause an error.

MQSO_ALTER

Return a handle to a pre-existing subscription with the full subscription name matching that specified by the name in *SubName*. Any attributes of the subscription that are different from that specified in the MQSD are altered in the subscription unless alteration is disallowed for that attribute. Details are noted in the description of each attribute and are summarized in the following table. If you try to alter an attribute that cannot be changed, or to alter a subscription that has set the MQSO_IMMUTABLE option, the call fails with the reason code shown in the following table.

The call fails with reason code MQRC_NO_SUBSCRIPTION if a subscription matching the full subscription name does not exist. You can avoid this failure by combining the MQSO_CREATE option with MQSO_ALTER.

Combining MQSO_CREATE with MQSO_ALTER returns a handle to a pre-existing subscription for the specified *SubName* if one is found; if there is no existing subscription, a new one is created using all the fields provided in the MQSD.

The user ID of the subscription is the user ID that created the subscription, or if it is later altered by a different user ID, it is the user ID of the most recent, successful alteration. If an AlternateUserId is used, and use of alternate user IDs is allowed for that user, then the alternate user ID is recorded as the user ID that created the subscription instead of the user ID under which the subscription was made.

If a matching subscription exists that was created without the option MQSO_ANY_USERID and the user ID of the subscription is different from that of the application requesting a handle to the subscription, the call fails with reason code MQRC_IDENTITY_MISMATCH.

If a matching subscription exists and is currently in use, the call fails with MQRC_SUBSCRIPTION_IN_USE.

If the subscription named in SubName is not a valid subscription to resume or alter from an application, the call fails with MQRC_INVALID_SUBSCRIPTION.

The following table shows the ability of MQSO_ALTER to alter attribute values in MQSD and MQSUB.

Table 194. Attributes in MQSD and MQSUB that can be altered

Data type descriptor or function call	Field name	Can this attribute be altered using MQSO_ALTER	Reason Code
MQSD	Durability options	No	MQRC_DURABILITY_NOT_ALTERABLE
MQSD	Destination Options	Yes	None
MQSD	Registration options	Yes (see note 1 on page 1866)	MQRC_GROUPING_NOT_ALTERABLE if you try to alter MQSO_GROUP_SUB
MQSD	Publication options	Yes (see note 2 on page 1866)	None
MQSD	Wildcard options	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	Other options	No (see note 3 on page 1866)	None
MQSD	ObjectName	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	AlternateUserId	No (see note 4 on page 1866)	None

Table 194. Attributes in MQSD and MQSUB that can be altered (continued)

Data type descriptor or function call	Field name	Can this attribute be altered using MQSO_ALTER	Reason Code
MQSD	AlternateSecurityId	No (see note 4)	None
MQSD	SubExpiry	Yes	None
MQSD	ObjectString	No	MQRC_TOPIC_NOT_ALTERABLE
MQSD	SubName	No (see note 5)	None
MQSD	SubUserData	Yes	None
MQSD	SubCorrelId	Yes (see note 6)	MQRC_GROUPING_NOT_ALTERABLE when in a grouped subscription
MQSD	PubPriority	Yes	None
MQSD	PubAccountingToken	Yes	None
MQSD	PubApplIdentityData	Yes	None
MQSD	SubLevel	No	MQRC_SUBLEVEL_NOT_ALTERABLE
MQSUB	Hobj	Yes (see note 6)	MQRC_GROUPING_NOT_ALTERABLE when in a grouped subscription

Notes:

1. MQSO_GROUP_SUB cannot be altered.
2. MQSO_NEW_PUBLICATIONS_ONLY cannot be altered because it is not part of the subscription
3. These options are not part of the subscription
4. This attribute is not part of the subscription
5. This attribute is the identity of the subscription being altered
6. Alterable except when part of a grouped sub (MQSO_GROUP_SUB)

Durability options: The following options control how durable the subscription is. You can specify only one of these options. If you are altering an existing subscription using the MQSO_ALTER option, you cannot change the durability of the subscription. On return from an MQSUB call using MQSO_RESUME, the appropriate durability option is set.

MQSO_DURABLE

Request that the subscription to this topic remains until it is explicitly removed using MQCLOSE with the MQCO_REMOVE_SUB option. If this subscription is not explicitly removed it will remain even after this applications connection to the queue manager is closed.

If a durable subscription is requested to a topic that is defined as not allowing durable subscriptions, the call fails with MQRC_DURABILITY_NOT_ALLOWED.

MQSO_NON_DURABLE

Request that the subscription to this topic is removed when the applications connection to the queue manager is closed, if it is not already explicitly removed. MQSO_NON_DURABLE is the opposite of the MQSO_DURABLE option, and is defined to aid program documentation. It is the default if neither is specified.

Destination options: The following option controls the destination that publications for a topic that has been subscribed to are sent to. If altering an existing subscription using the MQSO_ALTER option, the destination used for publications for the subscription can be changed. On return from an MQSUB call using MQSO_RESUME, this option is set if appropriate.

MQSO_MANAGED

Request that the destination that the publications are sent to is managed by the queue manager.

The object handle returned in *Hobj* represents a queue manager managed queue and is for use with subsequent MQGET, MQCB, MQINQ, or MQCLOSE calls.

An object handle returned from a previous MQSUB call cannot be provided in the *Hobj* parameter when MQSO_MANAGED is not specified.

MQSO_NO_MULTICAST

Request that the destination that the publications are sent to is not a multicast group address. This option is only valid when combined with the MQSO_MANAGED option. When a handle to a queue is provided in the *Hobj* parameter, multicast cannot be used for this subscription, and the option is not valid.

If the topic is defined to only allow multicast subscriptions, using the `MCAST(ONLY)` setting, then the call fails with reason code MQRC_MULTICAST_REQUIRED.

Scope Option: The following option controls the scope of the subscription being made. If altering an existing subscription using the MQSO_ALTER option, this subscription scope option cannot be changed. On returning from an MQSUB call using MQSO-RESUME, the appropriate scope option is set.

MQSO_SCOPE_QMGR

This subscription is made only on the local queue manager. No proxy subscription is distributed to other queue managers in the network. Only publications that are published at this queue manager are sent to this subscriber. This overrides any behavior set using the SUBSCOPE topic attribute.

Note: If not set, the subscription scope is determined by the SUBSCOPE topic attribute.

Registration options: The following options control the details of the registration that is made to the queue manager for this subscription. If altering an existing subscription using the MQSO_ALTER option, these registration options can be changed. On return from an MQSUB call using MQSO_RESUME the appropriate registration options is set.

MQSO_GROUP_SUB

This subscription is to be grouped with other subscriptions of the same SubLevel using the same queue and specifying the same correlation ID so that any publications to topics that would cause more than one publication message to be provided to the group of subscriptions, due to an overlapping set of topic strings being used, only causes one message to be delivered to the queue. If this option is not used, then each unique subscription (identified by SubName) that matches is provided with a copy of the publication which could mean more than one copy of the publication may be placed on the queue shared by a number of subscriptions.

Only the most significant subscription in the group is provided with a copy of the publication. The most significant subscription is based on the Full topic name up to the point where a wildcard is found. If a mixture of wildcard schemes is used within the group, only the position of the wildcard is important. You are advised not to combine different wildcard schemes within a group of subscriptions that share the same queue.

When creating a new grouped subscription it must still have a unique SubName, but if it matches the full topic name of an existing subscription in the group, the call fails with MQRC_DUPLICATE_GROUP_SUB.

If the most significant subscription in group also specifies MQSO_NOT_OWN_PUBS and this is a publication from the same application, then no publication is delivered to the queue.

When altering a subscription made with this option, the fields which imply the grouping, Hobj on the MQSUB call (representing the queue and queue manager name), and the SubCorrelId cannot be changed. Attempting to alter them causes the call to fail with MQRC_GROUPING_NOT_ALTERABLE.

This option must be combined with MQSO_SET_CORREL_ID with a SubCorrelId that is not set to MQCI_NONE, and cannot be combined with MQSO_MANAGED.

MQSO_ANY_USERID

When MQSO_ANY_USERID is specified, the identity of the subscriber is not restricted to a single user ID. This allows any user to alter or resume the subscription when they have suitable

authority. Only a single user may have the subscription at any one time. An attempt to resume use of a subscription currently in use by another application causes the call to fail with MQRC_SUBSCRIPTION_IN_USE.

To add this option to an existing subscription the MQSUB call (using MQSO_ALTER) must come from the same user ID as the original subscription itself.

If an MQSUB call refers to an existing subscription with MQSO_ANY_USERID set, and the user ID differs from the original subscription, the call succeeds only if the new user ID has authority to subscribe to the topic. On successful completion, future publications to this subscriber are put to the subscribers queue with the new user ID set in the publication message.

Do not specify both MQSO_ANY_USERID and MQSO_FIXED_USERID. If neither is specified, the default is MQSO_FIXED_USERID.

MQSO_FIXED_USERID

When MQSO_FIXED_USERID is specified, the subscription can be altered or resumed by only the last user ID to alter the subscription. If the subscription has not been altered, it is the user ID that created the subscription.

If an MQSUB verb refers to an existing subscription with MQSO_ANY_USERID set and alters the subscription using MQSO_ALTER to use option MQSO_FIXED_USERID, the user ID of the subscription is now fixed at this new user ID. The call succeeds only if the new user ID has authority to subscribe to the topic.

If a user ID other than the one recorded as owning a subscription tries to resume or alter an MQSO_FIXED_USERID subscription, the call fails with MQRC_IDENTITY_MISMATCH. The owning user ID of a subscription can be viewed using the DISPLAY SBSTATUS command.

Do not specify both MQSO_ANY_USERID and MQSO_FIXED_USERID. If neither is specified, the default is MQSO_FIXED_USERID.

Publication options: The following options control the way publications are sent to this subscriber. If altering an existing subscription using the MQSO_ALTER option, these publication options can be changed.

MQSO_NOT_OWN_PUBS

Tells the broker that the application does not want to see any of its own publications. Publications are considered to originate from the same application if the connection handles are the same. On return from an MQSUB call using MQSO_RESUME, this option is set if appropriate.

MQSO_NEW_PUBLICATIONS_ONLY

No currently retained publications are to be sent, when this subscription is created, only new publications. This option only applies when MQSO_CREATE is specified. Any subsequent changes to a subscription do not alter the flow of publications and so any publications retained on a topic, will have already been sent to the subscriber as new publications.

If this option is specified without MQSO_CREATE the call fails with MQRC_OPTIONS_ERROR. On return from an MQSUB call using MQSO_RESUME, this option is not set even if the subscription was created using this option.

If this option is not used, previously retained messages are sent to the destination queue provided. If this action fails due to an error, either MQRC_RETAINED_MSG_Q_ERROR or MQRC_RETAINED_NOT_DELIVERED, the creation of the subscription fails.

MQSO_PUBLICATIONS_ON_REQUEST

Setting this option indicates that the subscriber will request information specifically when required. The queue manager does not send unsolicited messages to the subscriber. The retained publication (or possibly multiple publications if a wildcard is specified in the topic) is sent to the subscriber each time an MQSUBRQ call is made using the Hsub handle from a previous MQSUB

call. No publications are sent as a result of the MQSUB call using this option. On return from an MQSUB call using MQSO_RESUME, this option is set if appropriate.

This option is not valid in combination with a SubLevel greater than 1.

Read ahead options: The following options control whether non-persistent messages are sent to an application ahead of the application requesting them.

MQSO_READ_AHEAD_AS_Q_DEF
If the MQSUB call uses a managed handle, the default read ahead attribute of the model queue associated with the topic subscribed to determines whether messages are sent to the application before the application requests them.

This is the default value.

MQSO_NO_READ_AHEAD
If the MQSUB call uses a managed handle, messages are not sent to the application before the application requests them.

MQSO_READ_AHEAD
If the MQSUB call uses a managed handle, messages might be sent to the application before the application requests them.

Note:

The following notes apply to the read ahead options:

1. Only one of these options can be specified. If both MQOO_READ_AHEAD and MQOO_NO_READ_AHEAD are specified, reason code MQRC_OPTIONS_ERROR is returned. These options are only applicable if MQSO_MANAGED is specified.
2. They are not applicable for MQSUB when a queue is passed which has been opened previously. Read ahead might not be enabled when requested. The MQGET options used on the first MQGET call might prevent read ahead from being enabled. Also, read ahead is disabled when the client is connecting to a queue manager where read ahead is not supported. If the application is not running as a WebSphere MQ client, these options are ignored.

Wildcard options: The following options control how wildcards are interpreted in the string provided in the ObjectString field of the MQSD. You can specify only one of these options. If altering an existing subscription using the MQSO_ALTER option, these wildcard options cannot be changed. On return from an MQSUB call using MQSO_RESUME, the appropriate wildcard option is set.

MQSO_WILDCARD_CHAR
Wildcards only operate on characters within the topic string.

The behavior defined by MQSO_WILDCARD_CHAR is shown in the following table.

Special Character	Behavior
Forward slash (/)	No significance, just another character
Asterisk (*)	Wildcard, zero or more characters
Question mark (?)	Wildcard, 1 character
Percent sign (%)	Escape character to allow the characters (*), (?) or (%) to be used in a string and not be interpreted as a special character, for example, (%*), (%?) or (%%).

For example, publishing on the following topic:

```
/level0/level1/level2/level3/level4
```

matches subscribers using the following topics:

```
*
/*
/ level0/level1/level2/level3/*
/ level0/level1/*/level3/level4
/ level0/level1/le?el2/level3/level4
```

Note: This use of wildcards supplies exactly the meaning provided in WebSphere MQ V6 and WebSphere MB V6 when using MQRFH1 formatted messages for publish/subscribe. It is recommended that this is not used for newly written applications and is only used for applications that were previously running against that version and have not been changed to use the default wildcard behavior as described in MQSO_WILDCARD_TOPIC.

MQSO_WILDCARD_TOPIC

Wildcards only operate on topic elements within the topic string. This is the default behavior if none is chosen.

The behavior required by MQSO_WILDCARD_TOPIC is shown in the following table:

Special Character	Behavior
(/)	Topic level separator
Number sign (#)	Wildcard: multiple topic level
Plus sign (+)	Wildcard: single topic level
Notes: The (+) and (#) are not treated as wildcards if they are mixed in with other characters (including themselves) within a topic level. In the following string, the (#) and (+) characters are treated as ordinary characters. `level0/level1/#+/level3/level#`	

For example, publishing on the following topic:

```
/level0/level1/level2/level3/level4
```

matches subscribers using the following topics:

```
#
/#
/ level0/level1/level2/level3/#
/ level0/level1/+/level3/level4
```

Note: This use of wildcards supplies the meaning provided in WebSphere Message Broker Version 6 when using MQRFH2 formatted messages for publish/subscribe.

Other options: The following options control the way the API call is issued rather than the subscription. On return from an MQSUB call using MQSO_RESUME, these options are unchanged. See "AlternateUserId (MQCHAR12)" on page 1861 for more details.

MQSO_ALTERNATE_USER_AUTHORITY

The AlternateUserId field contains a user identifier to use to validate this MQSUB call. The call can succeed only if this AlternateUserId is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.

MQSO_SET_CORREL_ID

The subscription is to use the correlation identifier supplied in the *SubCorrelId* field. If this option is not specified, a correlation identifier is automatically created by the queue manager at subscription time and is returned to the application in the *SubCorrelId* field. For more information, see "SubCorrelId (MQBYTE24)" on page 1874 for more information.

This option cannot be combined with MQSO_MANAGED.

MQSO_SET_IDENTITY_CONTEXT

The subscription is to use the accounting token and application identity data supplied in the *PubAccountingToken* and *PubApplIdentityData* fields.

If this option is specified, the same authorization check is carried out as if the destination queue was accessed using an MQOPEN call with MQOO_SET_IDENTITY_CONTEXT, except in the case where the MQSO_MANAGED option is also used in which case there is no authorization check on the destination queue.

If this option is not specified, the publications sent to this subscriber have default context information associated with them as follows:

Field in MQMD	Value used
UserIdentifier	The user ID associated with the subscription at the time the subscription was made.
AccountingToken	Determined from the environment if possible; Set to MQACT_NONE if not.
ApplIdentityData	Set to blanks

This option is only valid with MQSO_CREATE and MQSO_ALTER. If used with MQSO_RESUME, the *PubAccountingToken* and *PubApplIdentityData* fields are ignored, so this option has no effect.

If a subscription is altered without using this option where previously the subscription supplied identity context information, default context information is generated for the altered subscription.

If a subscription allowing different user IDs to use it with option MQSO_ANY_USERID, is resumed by a different user ID, default identity context is generated for the new user ID now owning the subscription and any subsequent publications are delivered containing the new identity context.

MQSO_FAIL_IF_QUIESCING

The MQSUB call fails if the queue manager is in quiescing state. On z/OS, for a CICS or IMS application, this option also forces the MQSUB call to fail if the connection is in quiescing state.

PubAccountingToken (MQBYTE32):

This is the value that will be in the *AccountingToken* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. *AccountingToken* is part of the identity context of the message. For more information about message context, see Message context. For more information about the *AccountingToken* field in the MQMD, see "AccountingToken (MQBYTE32)" on page 1708

You can use the following special value for the *PubAccountingToken* field:

MQACT_NONE

No accounting token is specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQACT_NONE_ARRAY is also defined; this has the same value as MQACT_NONE, but is an array of characters instead of a string.

If the option MQSO_SET_IDENTITY_CONTEXT is not specified, the accounting token is generated by the queue manager as default context information and this field is an output field which contains the *AccountingToken* which will be set in each message published for this subscription.

If the option MQSO_SET_IDENTITY_CONTEXT is specified, the accounting token is being generated by the user and this field is an input field which contains the *AccountingToken* to be set in each publication for this subscription.

The length of this field is given by MQ_ACCOUNTING_TOKEN_LENGTH. The initial value of this field is MQACT_NONE.

If altering an existing subscription using the MQSO_ALTER option, the value of *AccountingToken* in any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current *AccountingToken* being used for the subscription.

PubApplIdentityData (MQCHAR32):

This is the value that is in the *ApplIdentityData* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. *ApplIdentityData* is part of the identity context of the message. For more information about message context, see Message context. For more information about the *ApplIdentityData* field in the MQMD, see "ApplIdentityData (MQCHAR32)" on page 1710

If the option MQSO_SET_IDENTITY_CONTEXT is not specified, the *ApplIdentityData* which is set in each message published for this subscription is blanks, as default context information.

If the option MQSO_SET_IDENTITY_CONTEXT is specified, the *PubApplIdentityData* is being generated by the user and this field is an input field which contains the *ApplIdentityData* to be set in each publication for this subscription.

The length of this field is given by MQ_APPL_IDENTITY_DATA_LENGTH. The initial value of this field is the null string in C, and 32 blank characters in other programming languages.

If altering an existing subscription using the MQSO_ALTER option, the *ApplIdentityData* of any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current *ApplIdentityData* being used for the subscription.

PubPriority (MQLONG):

This is the value that will be in the *Priority* field of the Message Descriptor (MQMD) of all publication messages matching this subscription. For more information about the *Priority* field in the MQMD, see "Priority (MQLONG)" on page 1735.

The value must be greater than or equal to zero; zero is the lowest priority. The following special values can also be used:

MQPRI_PRIORITY_AS_Q_DEF

When a subscription queue is provided in the *Hobj* field in the MQSUB call, and is not a managed handle, then the priority for the message is taken from the *DefPriority* attribute of this queue. If the queue is a cluster queue or there is more than one definition in the queue-name resolution path then the priority is determined when the publication message is put to the queue as described for "Priority (MQLONG)" on page 1735.

If the MQSUB call uses a managed handle, the priority for the message is taken from the *DefPriority* attribute of the model queue associated with the topic subscribed to.

MQPRI_PRIORITY_AS_PUBLISHED

The priority for the message is the priority of the original publication. This is the initial value of the field.

If altering an existing subscription using the MQSO_ALTER option, the *Priority* of any future publication messages can be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current priority being used for the subscription.

ResObjectString (MQCHARV):

This is the long object name after the queue manager resolves the name provided in *ObjectName*.

If the long object name is provided in *ObjectString* and nothing is provided in *ObjectName*, then the value returned in this field is the same as provided in *ObjectString*.

If this field is omitted (that is ResObjectString.VSBufSize is zero) then the *ResObjectString* is not returned, but the length is returned in ResObjectString.VSLength. If the length is shorter than the full ResObjectString then it is truncated and returns as many of the rightmost characters as can fit in the provided length.

If *ResObjectString* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_RES_OBJECT_STRING_ERROR.

SelectionString (MQCHARV):

This is the string used to provide the selection criteria used when subscribing for messages from a topic.

This variable length field will be returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided, and also there is a positive buffer length in VSBufSize. If no buffer is provided on the call, only the length of the selection string will be returned in the VSLength field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only VSBufSize bytes are returned in the provided buffer.

If *SelectionString* is specified incorrectly, according to the description of how to use the "MQCHARV - Variable Length String" on page 1581 structure, or if it exceeds the maximum length, the call fails with reason code MQRC_SELECTION_STRING_ERROR.

SelectionString usage is described in Selectors.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQSD_STRUC_ID
Identifier for Subscription Descriptor structure.

For the C programming language, the constant MQSD_STRUC_ID_ARRAY is also defined; this has the same value as MQSD_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSD_STRUC_ID.

SubCorrelId (MQBYTE24):

This field contains a correlation identifier common to all publications matching this subscription.

Attention: a correlation identifier can only be passed between queue managers in a publish/subscribe cluster, not a hierarchy.

All publications sent to match this subscription contain this correlation identifier in the message descriptor. If multiple subscriptions get their publications from the same queue, using MQGET by correlation identifier allows only publications for a specific subscription to be obtained. This correlation identifier can either be generated by the queue manager or by the user.

If the option MQSO_SET_CORREL_ID is not specified, the correlation identifier is generated by the queue manager and this field is an output field containing the correlation identifier that will be set in each message published for this subscription. The generated correlation identifier consists of a 4-byte product identifier (AMQX or CSQM in either ASCII or EBCDIC) followed by a product specific implementation of a unique string.

If the option MQSO_SET_CORREL_ID is specified, the correlation identifier is generated by the user and this field is an input field containing the correlation identifier to be set in each publication for this subscription. In this case, if the field contains MQCI_NONE, the correlation identifier that is set in each message published for this subscription is the correlation identifier created by the original put of the message.

If the option MQSO_GROUP_SUB is specified and the correlation identifier specified is the same as an existing grouped subscription using the same queue and an overlapping topic string, only the most significant subscription in the group is provided with a copy of the publication.

The length of this field is given by MQ_CORREL_ID_LENGTH. The initial value of this field is MQCI_NONE.

If you are altering an existing subscription using the MQSO_ALTER option, and this field is an input field, then the subscription correlation identifier can be changed, unless the subscription is a grouped subscription, that is, it has been created using the option MQSO_GROUP_SUB, in which case the subscription correlation identifier cannot be changed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current correlation identifier for the subscription.

SubExpiry (MQLONG):

This is the time expressed in tenths of a second after which the subscription expires. No more publications will match this subscription after this interval has passed. As soon as a subscription expires, publications are no longer sent to the queue. However, the publications that are already there are not affected in any way. *SubExpiry* has no effect on publication expiry.

The following special value is recognized:

MQEI_UNLIMITED
The subscription has an unlimited expiration time.

If altering an existing subscription using the MQSO_ALTER option, the expiry of the subscription can be changed.

On return from an MQSUB call using the MQSO_RESUME option this field is set to the original expiry of the subscription and not the remaining expiry time.

SubLevel (MQLONG):

This is the level associated with the subscription. Publications are only delivered to this subscription if it is in the set of subscriptions with the highest SubLevel value less than or equal to the PubLevel used at publication time. However, if a publication has been retained, it is no longer available to subscribers at higher levels because it is republished at PubLevel 1.

The value must be in the range zero to 9. Zero is the lowest level.

The initial value of this field is 1.

For more information see Intercepting publications.

If altering an existing subscription using the MQSO_ALTER option, then the SubLevel cannot be changed.

Combining a SubLevel with a value greater than 1 with the option MQSO_PUBLICATIONS_ON_REQUEST is not allowed.

On return from an MQSUB call using MQSO_RESUME, this field is set to the current level being used for the subscription.

SubUserData (MQCHARV):

This specifies the subscription user data. The data provided on the subscription in this field will be included as the MQSubUserData message property of every publication sent to this subscription.

The maximum length of *SubUserData* is 10240.

If *SubUserData* is specified incorrectly, according to the description of how to use the MQCHARV structure, or if it exceeds the maximum length, the call fails with reason code MQRC_SUB_USER_DATA_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If altering an existing subscription using the MQSO_ALTER option, the subscription user data can be changed.

This variable length field is returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided and there is a positive buffer length in *VSBufLen*. If no buffer is provided on the call, only the length of the subscription user date is returned in the *VSLength* field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only *VSBufLen* bytes are returned in the provided buffer.

SubName (MQCHARV):

This specifies the subscription name. This field is only required if *Options* specifies the option MQSO_DURABLE, but if provided will be used by the queue manager for MQSO_NON_DURABLE as well.

If specified, *SubName* must be unique within the queue manager, because it is the method used to identify the subscription.

The maximum length of *SubName* is 10240.

This field serves two purposes. For an MQSO_DURABLE subscription, you use this field to identify a subscription so you can resume it after it has been created if you have either closed the handle to the subscription (using the MQCO_KEEP_SUB option) or have been disconnected from the queue manager. This is done using the MQSUB call with the MQSO_RESUME option. It is also displayed in the administrative view of subscriptions in the SUBNAME field in DISPLAY SBSTATUS.

If *SubName* is specified incorrectly, according to the description of how to use the MQCHARV structure, is left out when it is required (that is *SubName.VSLength* is zero), or if it exceeds the maximum length, the call fails with reason code MQRC_SUB_NAME_ERROR.

This is an input field. The initial values of the fields in this structure are the same as those in the MQCHARV structure.

If altering an existing subscription using the MQSO_ALTER option, the subscription name cannot be changed, because it is the identifying field used to find the referenced subscription. It is not changed on output from an MQSUB call with the MQSO_RESUME option.

Version (MQLONG):

This is the structure version number; the value must be:

MQSD_VERSION_1
Version-1 Subscription Descriptor structure.

The following constant specifies the version number of the current version:

MQSD_CURRENT_VERSION
Current version of Subscription Descriptor structure.

This is always an input field. The initial value of this field is MQSD_VERSION_1.

Using topic strings:

A topic is constructed from the subtopic identified in a topic object, and a subtopic provided by an application. You can use either subtopic as the topic name, or combine them to form a new topic name.

In an MQI program the full topic name is created by MQOPEN. It is composed of two fields used in publish/subscribe MQI calls, in the order listed:

1. The **TOPICSTR** attribute of the topic object, named in the **ObjectName** field.
2. The **ObjectString** parameter defining the subtopic provided by the application.

The resulting topic string is returned in the **ResObjectString** parameter.

These fields are considered to be present if the first character of each field is not a blank or null character, and the field length is greater than zero. If only one of the fields is present, it is used unchanged as the topic name. If neither field has a value, the call fails with reason code MQRC_UNKNOWN_OBJECT_NAME, or MQRC_TOPIC_STRING_ERROR if the full topic name is not valid.

If both fields are present, a '/' character is inserted between the two elements of the resultant combined topic name.

Table 195 on page 1877 shows examples of topic string concatenation:

Table 195. Topic string concatenation examples

TOPICSTR	ObjectString	Full topic name	Comment
Football/Scores	''	Football/Scores	The TOPICSTR is used alone
''	Football/Scores	Football/Scores	The ObjectString is used alone
Football	Scores	Football/Scores	A '/' character is added at the concatenation point
Football	/Scores	Football//Scores	An 'empty node' is produced between the two strings
/Football	Scores	/Football/Scores	The topic starts with an 'empty node'

The '/' character is considered as a special character, providing structure to the full topic name in Topic trees, and must not be used for any other reason as the structure of the topic tree is affected. The topic "`/Football`" is not the same as the topic "`Football`".

The following wildcard characters are special characters:

- plus sign '+'
- number sign '#'
- asterisk '*'
- question mark '?'

These characters are not considered as invalid, however you must ensure to understand how they are used. You might prefer not to use these characters in your topic strings when publishing. Publishing on a topic string with '#' or '+' mixed in with other characters (including themselves) within a topic level, can be subscribed to with either wildcard scheme. Publishing on a topic string with '#' or '+' as the only character between two '/' characters produces a topic string that cannot be subscribed to explicitly by an application using the wildcard scheme MQSO_WILDCARD_TOPIC. This situation results in the application getting more publications than expected.

Example code snippet

This code snippet, extracted from the example program Example 2: Publisher to a variable topic, combines a topic object with a variable topic string.

```
MQOD     td = {MQOD_DEFAULT};  /* Object Descriptor            */
td.ObjectType = MQOT_TOPIC;    /* Object is a topic            */
td.Version = MQOD_VERSION_4;   /* Descriptor needs to be V4    */
strncpy(td.ObjectName, topicName,  MQ_TOPIC_NAME_LENGTH);
td.ObjectString.VSPtr = topicString;
td.ObjectString.VSLength = (MQLONG)strlen(topicString);
td.ResObjectString.VSPtr = resTopicStr;
td.ResObjectString.VSBufSize = sizeof(resTopicStr)-1;
MQOPEN(Hconn, &td, MQOO_OUTPUT | MQOO_FAIL_IF_QUIESCING, &Hobj, &CompCode, &Reason);
```

Initial values and language declarations for MQSD:

Field name	Name of constant	Value of constant
StrucId	MQSD_STRUC_ID	'SD¬¬'
Version	MQSD_VERSION_1	1
Options	MQSO_NON_DURABLE	0
ObjectName	None	Null string or blanks
AlternateUserId	None	Null string or blanks
AlternateSecurityId	MQSID_NONE	Nulls
SubExpiry	MQEI_UNLIMITED	-1
ObjectString	None	Names and values as defined for MQCHARV
SubName	None	Names and values as defined for MQCHARV
SubUserData	None	Names and values as defined for MQCHARV
SubCorrelId	MQCI_NONE	Nulls
PubPriority	MQPRI_PRIORITY_AS_Q_DEF	-3
PubAccountingToken	MQACT_NONE	Nulls
PubApplIdentityData	None	Null string or blanks
Selection String	None	Names and values as defined for MQCHARV
SubLevel	None	1
ResObjectString	None	Names and values as defined for MQCHARV

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQSD_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

   ```
   MQSD MySD = {MQSD_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQSD MQSD;
struct tagMQSD {
  MQCHAR4   StrucId;              /* Structure identifier */
  MQLONG    Version;              /* Structure version number */
  MQLONG    Options;              /* Options associated with subscribing */
  MQCHAR48  ObjectName;           /* Object name */
  MQCHAR12  AlternateUserId;      /* Alternate user identifier */
  MQBYTE40  AlternateSecurityId;  /* Alternate security identifier */
  MQLONG    SubExpiry;            /* Expiry of Subscription */
  MQCHARV   ObjectString;         /* Object Long name */
  MQCHARV   SubName;              /* Subscription name */
  MQCHARV   SubUserData;          /* Subscription User data */
  MQBYTE24  SubCorrelId;          /* Correlation Id related to this subscription */
  MQLONG    PubPriority;          /* Priority set in publications */
  MQBYTE32  PubAccountingToken;   /* Accounting Token set in publications */
  MQCHAR32  PubApplIdentityData;  /* Appl Identity Data set in publications */
```

```
  MQCHARV   SelectionString;      /* Message selector structure */
  MQLONG    SubLevel;             /* Subscription level */
  MQCHARV   ResObjectString;      /* Resolved Long object name*/
  /* Ver:1 */
};
```

COBOL declaration:

```
** Address of variable length string
   20  MQSD-OBJECTSTRING-VSPTR         POINTER.
** Offset of variable length string
   20  MQSD-OBJECTSTRING-VSOFFSET      PIC S9(9) BINARY.
** size of buffer
   20  MQSD-OBJECTSTRING-VSBUFSIZE      PIC S9(9) BINARY.
** Length of variable length string
   20  MQSD-OBJECTSTRING-VSLENGTH      PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQSD-OBJECTSTRING-VSCCSID       PIC S9(9) BINARY.
** Subscription name
 15  MQSD-SUBNAME.
** Address of variable length string
   20  MQSD-SUBNAME-VSPTR              POINTER.
** Offset of variable length string
   20  MQSD-SUBNAME-VSOFFSET           PIC S9(9) BINARY.
** size of buffer
   20  MQSD-SUBNAME-VSBUFSIZE           PIC S9(9) BINARY.
** Length of variable length string
   20  MQSD-SUBNAME-VSLENGTH           PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQSD-SUBNAME-VSCCSID            PIC S9(9) BINARY.
** Subscription User data
 15  MQSD-SUBUSERDATA.
** Address of variable length string
   20  MQSD-SUBUSERDATA-VSPTR          POINTER.
** Offset of variable length string
   20  MQSD-SUBUSERDATA-VSOFFSET       PIC S9(9) BINARY.
** size of buffer
   20  MQSD-SUBUSERDATA-VSBUFSIZE       PIC S9(9) BINARY.
** Length of variable length string
   20  MQSD-SUBUSERDATA-VSLENGTH       PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQSD-SUBUSERDATA-VSCCSID        PIC S9(9) BINARY.
** Correlation Id related to this subscription
 15  MQSD-SUBCORRELID                 PIC X(24).
** Priority set in publications
 15  MQSD-PUBPRIORITY                 PIC S9(9) BINARY.
** Accounting Token set in publications
 15  MQSD-PUBACCOUNTINGTOKEN          PIC X(32).
** Appl Identity Data set in publications
 15  MQSD-PUBAPPLIDENTITYDATA         PIC X(32).
** Message Selector
 15 MQSD-SELECTIONSTRING.
** Address of variable length string
   20  MQSD-SELECTIONSTRING-VSPTR     POINTER.
** Offset of variable length string
   20  MQSD-SELECTIONSTRING-VSOFFSET       PIC S9(9) BINARY.
** size of buffer
   20  MQSD-SELECTIONSTRING-VSBUFSIZE      PIC S9(9) BINARY.
** Length of variable length string
   20  MQSD-SELECTIONSTRING-VSLENGTH       PIC S9(9) BINARY.
** CCSID of variable length string
   20  MQSD-SELECTIONSTRING-VSCCSID        PIC S9(9) BINARY.
** Selection criteria
   20  MQSD-SELECTIONSTRING-SUBLEVEL       PIC S9(9) BINARY.
** Long object name
   20  MQSD-SELECTIONSTRING-RESOBJSTRING   PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQSD based,
  3 StrucId             char(4),       /* Structure identifier */
  3 Version             fixed bin(31), /* Structure version number */
  3 Options             fixed bin(31), /* Options associated with subscribing */
  3 ObjectName          char(48),      /* Object name */
  3 AlternateUserId     char(12),      /* Alternate user identifier */
  3 AlternateSecurityId char(40),      /* Alternate security identifier */
  3 SubExpiry           fixed bin(31), /* Expiry of Subscription */
  3 ObjectString,                      /* Object Long name */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31); /* CCSID of variable length string */
  3 SubName,                           /* Subscription name */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31); /* CCSID of variable length string */
  3 SubUserData,                       /* Subscription User data */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31), /* CCSID of variable length string */
  3 SubCorrelId         char(24),      /* Correlation Id related to this subscription */
  3 PubPriority         fixed bin(31), /* Priority set in publications */
  3 PubAccountingToken  char(32),      /* Accounting Token set in publications */
  3 PubApplIdentityData char(32),      /* Appl Identity Data set in publications */
  3 SelectionString,                   /* Message Selection */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31), /* CCSID of variable length string */
  3 SubLevel            fixed bin(31), /* Subscription level */
  3 ResObjectString,                   /* Resolved Long object name */
   5 VSPtr              pointer,       /* Address of variable length string */
   5 VSOffset           fixed bin(31), /* Offset of variable length string */
   5 VSBufSize          fixed bin(31), /* size of buffer */
   5 VSLength           fixed bin(31), /* Length of variable length string */
   5 VSCCSID            fixed bin(31); /* CCSID of variable length string */
```

High Level Assembler declaration:

```
MQSD                          DSECT
MQSD_STRUCID                  DS    CL4   Structure identifier
MQSD_VERSION                  DS    F     Structure version number
MQSD-OPTIONS                  DS    F     Options associated with subscribing
MQSD_OBJECTNAME               DS    CL48  Object name
MQSD_ALTERNATEUSERID          DS    CL12  Alternate user identifier
MQSD_ALTERNATESECURITYID      DS    CL40  Alternate security identifier
MQSD_SUBEXPIRY                DS    F     Expiry of Subscription
MQSD_OBJECTSTRING             DS    0F    Object Long name
MQSD_OBJECTSTRING_VSPTR       DS    F     Address of variable length string
MQSD_OBJECTSTRING_VSOFFSET    DS    F     Offset of variable length string
MQSD_OBJECTSTRING_VSBUFSIZE   DS    F     size of buffer
MQSD_OBJECTSTRING_VSLENGTH    DS    F     Length of variable length string
MQSD_OBJECTSTRING_VSCCSID     DS    F     CCSID of variable length string
MQSD_OBJECTSTRING_LENGTH      EQU   *-MQSD_OBJECTSTRING
                              ORG   MQSD_OBJECTSTRING
MQSD_OBJECTSTRING_AREA        DS    CL(MQSD_OBJECTSTRING_LENGTH)
*
MQSD_SUBNAME                  DS    0F    Subscription name
MQSD_SUBNAME_VSPTR            DS    F     Address of variable length string
MQSD_SUBNAME_VSOFFSET         DS    F     Offset of variable length string
MQSD_SUBNAME_VSBUFSIZE        DS    F     size of buffer
MQSD_SUBNAME_VSLENGTH         DS    F     Length of variable length string
MQSD_SUBNAME_VSCCSID          DS    F     CCSID of variable length string
MQSD_SUBNAME_LENGTH           EQU   *-MQSD_SUBNAME
                              ORG   MQSD_SUBNAME
```

```
MQSD_SUBNAME_AREA              DS    CL(MQSD_SUBNAME_LENGTH)
*
MQSD_SUBUSERDATA               DS    0F    Subscription User data
MQSD_SUBUSERDATA_VSPTR         DS    F     Address of variable length string
MQSD_SUBUSERDATA_VSOFFSET      DS    F     Offset of variable length string
MQSD_SUBUSERDATA_VSBUFSIZE     DS    F     size of buffer
MQSD_SUBUSERDATA_VSLENGTH      DS    F     Length of variable length string
MQSD_SUBUSERDATA_VSCCSID       DS    F     CCSID of variable length string
MQSD_SUBUSERDATA_LENGTH        EQU   *-MQSD_SUBUSERDATA
                               ORG   MQSD_SUBUSERDATA
MQSD_SUBUSERDATA_AREA          DS    CL(MQSD_SUBUSERDATA_LENGTH)
*
MQSD_SUBCORRELID               DS    CL24  Correlation Id related to this subscription
MQSD_PUBPRIORITY               DS    F     Priority set in publications
MQSD_PUBACCOUNTINGTOKEN        DS    CL32  Accounting Token set in publications
MQSD_PUBAPPLIDENTITYDATA       DS    CL32  Appl Identity Data set in publications
*
MQSD_SELECTIONSTRING           DS   F     Message Selector
MQSD_SELECTIONSTRING_VSPTR     DS   F     Address of variable length string
MQSD_SELECTIONSTRING_VSOFFSET  DS   F     Offset of variable length string
MQSD_SELECTIONSTRING_VSBUFSIZE DS   F     size of buffer
MQSD_SELECTIONSTRING_VSLENGTH  DS   F     Length of variable length string
MQSD_SELECTIONSTRING_VSCCSID   DS   F     CCSID of variable length string
MQSD_SELECTIONSTRING_LENGTH    EQU  *- MQSD_SELECTIONSTRING
                               ORG  MQSD_SELECTIONSTRING
MQSD_SELECTIONSTRING_AREA      DS   CL(MQSD_SELECTIONSTRING_LENGTH)
*
MQSD-SUBLEVEL                  DS   F     Subscription level
*
MQSD_RESOBJECTSTRING           DS   F     Resolved Long object name
MQSD_RESOBJECTSTRING_VSPTR     DS   F     Address of variable length string
MQSD_RESOBJECTSTRING_VSOFFSET  DS   F     Offset of variable length string
MQSD_RESOBJECTSTRING_VSBUFSIZE DS   F     size of buffer
MQSD_RESOBJECTSTRING_VSLENGTH  DS   F     Length of variable length string
MQSD_RESOBJECTSTRING_VSCCSID   DS   F     CCSID of variable length string
MQSD_RESOBJECTSTRING_LENGTH    EQU  *- MQSD_RESOBJECTSTRING
                               ORG  MQSD_RESOBJECTSTRING
MQSD_RESOBJECTSTRING_AREA      DS   CL(MQSD_RESOBJECTSTRING_LENGTH)
*
MQSD_LENGTH                    EQU   *-MQSD
                               ORG   MQSD
MQSD_AREA                      DS    CL(MQSD_LENGTH)
```

MQSMPO - Set message property options:

The following table summarizes the fields in the structure.

Table 196. Fields in MQSMPO

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options
ValueEncoding	Property value encoding	ValueEncoding
ValueCCSID	Property value character set	ValueCCSID

Overview for MQSMPO:

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The **MQSMPO** structure allows applications to specify options that control how properties of messages are set. The structure is an input parameter on the **MQSETMP** call.

Character set and encoding: Data in **MQSMPO** must be in the character set of the application and encoding of the application (**MQENC_NATIVE**).

Fields for MQSMPO:

The MQSMPO structure contains the following fields; the fields are described in **alphabetical order**:

Options (MQLONG):

Location options: The following options relate to the relative location of the property compared to the property cursor:

MQSMPO_SET_FIRST
Sets the value of the first property that matches the specified name, or if it does not exist, adds a new property after all other properties with a matching hierarchy.

MQSMPO_SET_PROP_UNDER_CURSOR
Sets the value of the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused on an MQGET call, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

MQSMPO_SET_PROP_BEFORE_CURSOR
Sets a new property before the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused on an MQGET call, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

MQSMPO_SET_PROP_AFTER_CURSOR
Sets a new property after the property pointed to by the property cursor. The property pointed to by the property cursor is the one that was last inquired using either the MQIMPO_INQ_FIRST or the MQIMPO_INQ_NEXT option.

The property cursor is reset when the message handle is reused on an MQGET call, or when the message handle is specified in the *MsgHandle* field of the MQGMO or MQPMO structure on an MQPUT call.

If this option is used when the property cursor has not yet been established or if the property pointer to by the property cursor has been deleted, the call fails with completion code MQCC_FAILED and reason code MQRC_PROPERTY_NOT_AVAILABLE.

If you need none of the options described, use the following option:

MQSMPO_NONE
No options specified.

This is always an input field. The initial value of this field is MQSMPO_SET_FIRST.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQSMPO_STRUC_ID

Identifier for set message property options structure.

For the C programming language, the constant **MQSMPO_STRUC_ID_ARRAY** is also defined; this has the same value as **MQSMPO_STRUC_ID**, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is **MQSMPO_STRUC_ID**.

ValueCCSID (MQLONG):

The character set of the property value to be set if the value is a character string.

This is always an input field. The initial value of this field is **MQCCSI_APPL**.

ValueEncoding (MQLONG):

The encoding of the property value to be set if the value is numeric.

This is always an input field. The initial value of this field is **MQENC_NATIVE**.

Version (MQLONG):

This is the structure version number; the value must be:

MQSMPO_VERSION_1
Version-1 set message property options structure.

The following constant specifies the version number of the current version:

MQSMPO_CURRENT_VERSION
Current version of set message property options structure.

This is always an input field. The initial value of this field is **MQSMPO_VERSION_1**.

Initial values and language declarations for MQSMPO:

Table 197. Initial values of fields in MQSMPO

Field name	Name of constant	Value of constant
StrucId	MQSMPO_STRUC_ID	'SMPO'
Version	MQSMPO_VERSION_1	1
Options	MQSMPO_NONE	0
ValueEncoding	MQENC_NATIVE	Depends on environment
ValueCCSID	MQCCSI_APPL	-3

Table 197. Initial values of fields in MQSMPO (continued)

Field name	Name of constant	Value of constant
Notes: 1. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages. 2. In the C programming language, the macro variable MQSMPO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure: `MQSMPO MySMPO = {MQSMPO_DEFAULT};`		

C declaration:

```
typedef struct tagMQSMPO MQSMPO;
struct tagMQSMPO {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    Options;         /* Options that control the action of MQSETMP */
  MQLONG    ValueEncoding;   /* Encoding of Value */
  MQLONG    ValueCCSID;      /* Character set identifier of Value */
};
```

COBOL declaration:

```
**   MQSMPO structure
  10 MQSMPO.
**    Structure identifier
   15 MQSMPO-STRUCID       PIC X(4).
**    Structure version number
   15 MQSMPO-VERSION       PIC S9(9) BINARY.
**    Options that control the action of MQSETMP
   15 MQSMPO-OPTIONS       PIC S9(9) BINARY.
**    Encoding of VALUE
   15 MQSMPO-VALUEENCODING PIC S9(9) BINARY.
**    Character set identifier of VALUE
   15 MQSMPO-VALUECCSID    PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQSMPO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of MQSETMP */
  3 ValueEncoding  fixed bin(31), /* Encoding of Value */
  3 ValueCCSID     fixed bin(31), /* Character set identifier of Value */
```

High Level Assembler declaration:

```
MQSMPO                DSECT
MQSMPO_STRUCID        DS   CL4   Structure identifier
MQSMPO_VERSION        DS   F     Structure version number
MQSMPO_OPTIONS        DS   F     Options that control the action of
*                                MQSETMP
MQSMPO_VALUEENCODING  DS   F     Encoding of VALUE
MQSMPO_VALUECCSID     DS   F     Character set identifier of VALUE
MQSMPO_LENGTH         EQU  *-MQSMPO
MQSMPO_AREA           DS   CL(MQSMPO_LENGTH)
```

MQSRO - Subscription request options:

This section describes subscription request options, what fields it contains, and initial values of those fields.

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
Options	Options	Options
NumPubs	Number of publications	NumPubs

Overview for MQSRO:
Availability: AIX, HP-UX, IBM i, Solaris, Linux, Windows, z/OS plus WebSphere MQ MQI clients connected to these systems.

Purpose: The MQSRO structure allows the application to specify options that control how a subscription request is made. The structure is an input/output parameter on the MQSUBRQ call.

Version: The current version of MQSRO is MQSRO_VERSION_1.

Character set and encoding: Data in MQSRO must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE. However, if the application is running as an MQ MQI client, the structure must be in the character set and encoding of the client.

Fields for MQSRO:

The MQSRO structure contains the following fields; the fields are described in alphabetical order:

NumPubs (MQLONG):

This is an output field, returned to the application to indicate the number of publications sent to the subscription queue as a result of this call. Although this number of publications have been sent as a result of this call, there is no guarantee that this many messages will be available for the application to get, especially if they are non-persistent messages.

There might be more than one publication if the topic subscribed to contained a wildcard. If no wildcards were present in the topic string when the subscription represented by *Hsub* was created, then at most one publication is sent as a result of this call.

Options (MQLONG):

One of the following options must be specified. Only one option can be specified.

MQSRO_FAIL_IF_QUIESCING

The MQSUBRQ call fails if the queue manager is in the quiescing state. On z/OS, for a CICS or IMS application, this option also forces the MQSUBRQ call to fail if the connection is in a quiescing state.

Default option: If the option described above is not required, the following option must be used:

MQSRO_NONE

Use this value to indicate that no other options have been specified; all options assume their default values.

MQSRO_NONE helps program documentation. Although it is not intended that this option be used with any other, because its value is zero, this use cannot be detected.

StrucId (MQCHAR4):

This is the structure identifier; the value must be:

MQSRO_STRUC_ID
Identifier for Subscription Request Options structure.

For the C programming language, the constant MQSRO_STRUC_ID_ARRAY is also defined; this has the same value as MQSRO_STRUC_ID, but is an array of characters instead of a string.

This is always an input field. The initial value of this field is MQSRO_STRUC_ID.

Version (MQLONG):

This is the structure version number; the value must be:

MQSRO_VERSION_1
Version-1 Subscription Request Options structure.

The following constant specifies the version number of the current version:

MQSRO_CURRENT_VERSION
Current version of Subscription Request Options structure.

This is always an input field. The initial value of this field is MQSRO_VERSION_1.

Initial values and language declarations for MQSRO:

Field name	Name of constant	Value of constant
StrucId	MQSRO_STRUC_ID	'SRO¬'
Version	MQSRO_VERSION_1	1
Options	MQSRO_NONE	0
NumPubs	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQSRO_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQSRO MySRO = {MQSRO_DEFAULT};
```

C declaration:

```
typedef struct tagMQSRO MQSRO;
struct tagMQSRO {
  MQCHAR4   StrucId;              /* Structure identifier */
  MQLONG    Version;              /* Structure version number */
  MQLONG    Options;              /* Options that control the action of MQSUBRQ */
  MQLONG    NumPubs;              /* Number of publications sent */
  /* Ver:1 */
};
```

COBOL declaration:

```
** MQSRO structure
 10  MQSRO.
** Structure identifier
 15  MQSRO-STRUCID             PIC X(4).
** Structure version number
 15  MQSRO-VERSION             PIC S9(9) BINARY.
** Options that control the action of MQSUBRQ
 15  MQSRO-OPTIONS             PIC S9(9) BINARY.
** Number of publications sent
 15  MQSRO-NUMPUBS             PIC S9(9) BINARY.
```

PL/I declaration:

```
dcl
 1 MQSRO based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 Options        fixed bin(31), /* Options that control the action of MQSUBRQ */
  3 NumPubs        fixed bin(31); /* Number of publications sent */
```

High Level Assembler declaration:

```
MQSRO                   DSECT
MQSRO_STRUCID           DS    CL4   Structure identifier
MQSRO_VERSION           DS    F     Structure version number
MQSRO_OPTIONS           DS    F     Options that control the action of MQSUBRQ
MQSRO_NUMPUBS           DS    F     Number of publications sent
*
MQSRO_LENGTH            EQU   *-MQSRO
                        ORG   MQSRO
MQSRO_AREA              DS    CL(MQSRO_LENGTH)
```

MQSTS - Status reporting structure:

The following table summarizes the fields in the structure.

Table 198. Fields in MQSTS

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
CompCode	Completion code of first error	CompCode
Reason	Reason code of first error	Reason
PutSuccessCount	Number of successful asynchronous put calls	SuccessCount
PutWarningcount	Number of asynchronous put calls which had warnings	WarningCount
PutFailureCount	Number of failed asynchronous put calls	FailureCount
ObjectType	Type of failing object	ObjectType
ObjectName	Name of failing object	ObjectName
ObjectQMgrName	Name of queue manager owning the failing object	ObjectQMgrName
ResolvedObjectName	Resolved name of destination queue	ResolvedObjectName
ResolvedQMgrName	Resolved name of destination queue manager	ResolvedQMgrName
Note: The remaining fields are ignored if Version is less than MQSTS_VERSION_2.		
ObjectString	Long object name of failing object	ObjectString
SubName	Subscription name of failing subscription	SubName
OpenOptions	Open options associated with the failure	OpenOptions

Table 198. Fields in MQSTS (continued)

Field	Description	Topic
SubOptions	Subscription options associated with the failure	SubOptions

Overview for MQSTS:

Purpose: The MQSTS structure is an output parameter from the MQSTAT command.

Character set and encoding: Character data in MQSTS is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQSTS is in the native machine encoding; this is given by *Encoding*.

Usage: The MQSTAT command is used to retrieve status information. This information is returned in an MQSTS structure. For information about MQSTAT, see "MQSTAT - Retrieve status information" on page 2082.

Fields for MQSTS:

The MQSTS structure contains the following fields; the fields are described in **alphabetical order**:

CompCode (MQLONG):

The completion code of the operation being reported on.

The interpretation of `CompCode` depends on the value of the `MQSTAT Type` parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the completion code resulting from a previous asynchronous put operation on the object specified in `ObjectName`.

MQSTAT_TYPE_RECONNECTION

If the connection is reconnecting or failed to reconnect this is the completion code that caused the connection to begin reconnecting.

If the connection is currently connected the value is `MQCC_OK`.

MQSTAT_TYPE_RECONNECTION_ERROR

If the connection failed to reconnect this is the completion code that caused the reconnection to fail.

If the connection is currently connected, or reconnecting, the value is `MQCC_OK`.

`CompCode` is always an output field. Its initial value is `MQCC_OK`.

ObjectName (MQCHAR48):

The name of the object being reported on.

The interpretation of `ObjectName` depends on the value of the `MQSTAT Type` parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the name of the queue or topic used in the put operation, the failure of which is reported in the *CompCode* and *Reason* fields in the `MQSTS` structure.

MQSTAT_TYPE_RECONNECTION

If the connection is reconnecting, this is the name of the queue manager associated with the connection.

MQSTAT_TYPE_RECONNECTION_ERROR

If the connection failed to reconnect, this is the name of the object which caused reconnection to fail. The reason for the failure is reported in the *CompCode* and *Reason* fields in the MQSTS structure.

ObjectName is an output field. Its initial value is the null string in C, and 48 blank characters in other programming languages.

ObjectQMgrName (MQCHAR48):

The name of the queue manager being reported on.

The interpretation of ObjectQMgrName depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the name of the queue manager on which the *ObjectName* object is defined. A name that is entirely blank up to the first null character or the end of the field denotes the queue manager to which the application is connected (the local queue manager).

MQSTAT_TYPE_RECONNECTION

Blank.

MQSTAT_TYPE_RECONNECTION_ERROR

If the connection failed to reconnect, this is the name of the object which caused reconnection to fail. The reason for the failure is reported in the *CompCode* and *Reason* fields in the MQSTS structure.

ObjectQMgrName is an output field. Its value is the null string in C, and 48 blank characters in other programming languages.

ObjectString (MQCHARV):

Long object name of failing object being reported on. Present only in Version 2 of MQSTS or higher.

The interpretation of ObjectString depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the long object name of the queue or topic used in the MQPUT operation, which failed.

MQSTAT_TYPE_RECONNECTION

Zero length string

MQSTAT_TYPE_RECONNECTION_ERROR

This is the long object name of the object that caused the reconnection to fail.

ObjectString is an output field. Its initial value is a zero length string.

ObjectType (MQLONG):

The type of the object named in *ObjectName* being reported on.

Possible values of ObjectType are listed in "MQOT_* (Object Types and Extended Object Types)" on page 1470.

ObjectType is an output field. Its initial value is MQOT_Q.

OpenOptions (MQLONG):

The OpenOptions used to open the object being reported upon. Present only in Version 2 of MQSTS or higher.

The value of OpenOptions depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

Zero.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

The OpenOptions used when the failure occurred. The reason for the failure is reported in the *CompCode* and *Reason* fields in the MQSTS structure.

OpenOptions is an output field. Its initial value is zero.

PutFailureCount (MQLONG):

The number of asynchronous put operations that failed.

The value of PutFailureCount depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

The number of asynchronous put operations to the object named in the MQSTS structure that completed with MQCC_FAILED.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

Zero.

PutFailureCount is an output field. Its initial value is zero.

PutSuccessCount (MQLONG):

The number of asynchronous put operations that succeeded.

The value of PutSuccessCount depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

The number of asynchronous put operations to the object named in the MQSTS structure that completed with MQCC_OK.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

Zero.

PutSuccessCount is an output field. Its initial value is zero.

PutWarningCount (MQLONG):

The number of asynchronous put operations that ended with a warning.

The value of PutWarningCount depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

The number of asynchronous put operations to the object named in the MQSTS structure that completed with MQCC_WARNING.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

Zero.

PutWarningCount is an output field. Its initial value is zero.

SubName (MQCHARV):

The name of the failing subscription. Present only in Version 2 of MQSTS or higher.

The interpretation of SubName depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

Zero length string.

MQSTAT_TYPE_RECONNECTION

Zero length string.

MQSTAT_TYPE_RECONNECTION_ERROR

The name of the subscription that caused reconnection to fail. If no subscription name is available, or the failure is not related to a subscription, this is a zero-length string.

SubName is an output field. Its initial value is a zero length string.

SubOptions (MQLONG):

The SubOptions used to open the failing subscription. Present only in Version 2 of MQSTS or higher.

The interpretation of SubOptions depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

Zero.

MQSTAT_TYPE_RECONNECTION

Zero.

MQSTAT_TYPE_RECONNECTION_ERROR

The SubOptions used when the failure occurred. If the failure is not related to subscribing to a topic, the value returned is zero.

SubOptions is an output field. Its initial value is zero.

Reason (MQLONG):

The reason code of the operation being reported on.

The interpretation of Reason depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

This is the reason code resulting from a previous asynchronous put operation on the object specified in ObjectName.

MQSTAT_TYPE_RECONNECTION

If the connection is reconnecting or failed to reconnect this is the reason code that caused the reconnection to begin reconnecting.

If the connection is currently connected the value is MQRC_NONE.

MQSTAT_TYPE_RECONNECTION_ERROR

If the connection failed to reconnect this is the reason code that caused the reconnection to fail.

If the connection is currently connected, or reconnecting, the value is MQRC_NONE.

Reason is an output field. Its initial value is MQRC_NONE.

ResolvedObjectName (MQCHAR48):

The name of the object named in *ObjectName* after the local queue manager resolves the name.

The interpretation of ResolvedObjectName depends on the value of the MQSTAT Type parameter.

MQSTAT_TYPE_ASYNC_ERROR

ResolvedObjectName is the name of the object named in *ObjectName* after the local queue manager resolves the name. The name returned is the name of an object that exists on the queue manager identified by *ResolvedQMgrName*.

MQSTAT_TYPE_RECONNECTION

Blank.

MQSTAT_TYPE_RECONNECTION_ERROR

Blank.

ResolvedObjectName is an output field. Its initial value is the null string in C, and 48 blank characters in other programming languages.

ResolvedQMgrName (MQCHAR48):

The name of the destination queue manager after the local queue manager resolves the name.

The interpretation of `ResolvedQMgrName` depends on the value of the `MQSTAT Type` parameter.

MQSTAT_TYPE_ASYNC_ERROR

`ResolvedQMgrName` is the name of the destination queue manager after the local queue manager resolves the name. The name returned is the name of the queue manager that owns the object identified by *`ResolvedObjectName`*. *`ResolvedQMgrName`* might be the name of the local queue manager.

MQSTAT_TYPE_RECONNECTION

Blank.

MQSTAT_TYPE_RECONNECTION_ERROR

Blank.

`ResolvedQMgrName` is always an output field. Its initial value is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4):

The identifier for the status reporting structure, `MQSTS`.

`StrucId` is the structure identifier. The value must be:

MQSTS_STRUC_ID
Identifier for status reporting structure.

For the C programming language, the constant `MQSTS_STRUC_ID_ARRAY` is also defined; this has the same value as `MQSTS_STRUC_ID`, but is an array of characters instead of a string.

`StrucId` is always an input field. Its initial value is `MQSTS_STRUC_ID`.

Version (MQLONG):

The structure version number.

The value must be either:

MQSTS_VERSION_1
Version 1 status reporting structure.

MQSTS_VERSION_2
Version 2 status reporting structure.

The following constant specifies the version number of the current version:

MQSTS_CURRENT_VERSION
Current version of status reporting structure. The current version is `MQSTS_VERSION_2`.

`Version` is always an input field. Its initial value is `MQSTS_VERSION_1`.

Initial values and language declarations for MQSTS:

Table 199. Initial values of fields in MQSTS

Field name	Name of constant	Value of constant
StrucId	MQSTS_STRUC_ID	'STAT¬'
Version	MQSTS_VERSION_1	1
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
PutSuccessCount	None	0
PutWarningCount	None	0
PutFailureCount	None	0
ObjectType	MQOT_Q	1
ObjectName	None	Null string or blanks
ObjectQMgrName	None	Null string or blanks
ResolvedObjectName	None	Null string or blanks
ResolvedQMgrName	None	Null string or blanks
ObjectString	MQCHARV_DEFAULT	{NULL,0,0,0,-3}
SubName	MQCHARV_DEFAULT	{NULL,0,0,0,-3}
OpenOptions	None	0
SubOptions	None	0

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQSTS_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

   ```
   MQSTS MySTS = {MQSTS_DEFAULT};
   ```

C declaration:

```
typedef struct tagMQSTS MQSTS;
struct tagMQSTS {
 MQCHAR4   StrucId;            /* Structure identifier */
 MQLONG    Version;            /* Structure version number */
 MQLONG    CompCode;           /* Completion Code of first error */
 MQLONG    Reason;             /* Reason Code of first error */
 MQLONG    PutSuccessCount;    /* Number of Async calls succeeded */
 MQLONG    PutWarningCount;    /* Number of Async calls had warnings */
 MQLONG    PutFailureCount;    /* Number of Async calls had failures */
 MQLONG    ObjectType;         /* Failing object type */
 MQCHAR48  ObjectName;         /* Failing object name */
 MQCHAR48  ObjectQMgrName;     /* Failing object queue manager name */
 MQCHAR48  ResolvedObjectName; /* Resolved name of destination queue */
 MQCHAR48  ResolvedQMgrName;   /* Resolved name of destination qmgr */
/* Ver:1 */
 MQCHARV   ObjectString;       /* Failing object long name */
 MQCHARV   SubName;            /* Failing subscription name */
 MQLONG    OpenOptions;        /* Failing open options */
 MQLONG    SubOptions;         /* Failing subscription options */
/* Ver:2 */
};
```

COBOL declaration:

```
** MQSTS structure
 10  MQSTS.
** Structure identifier
 15  MQSTS-STRUCID PIC X(4).
** Structure version number
 15  MQSTS-VERSION PIC S9(9) BINARY.
** Completion Code of first error
 15  MQSTS-COMPCODE PIC S9(9) BINARY.
** Reason Code of first error
 15  MQSTS-REASON PIC S9(9) BINARY.
** Number of Async put calls succeeded
 15  MQSTS-PUTSUCCESSCOUNT PIC S9(9) BINARY.
** Number of Async put calls had warnings
 15  MQSTS-PUTWARNINGCOUNT PIC S9(9) BINARY.
** Number of Async put calls had failures
 15  MQSTS-PUTFAILURECOUNT PIC S9(9) BINARY.
** Failing object type
 15  MQSTS-OBJECTTYPE PIC S9(9) BINARY.
** Failing object name
 15  MQSTS-OBJECTNAME PIC X(48).
** Failing object queue manager
 15  MQSTS-OBJECTQMGRNAME PIC X(48).
** Resolved name of destination queue
 15  MQSTS-RESOLVEDOBJECTNAME PIC X(48).
** Resolved name of destination qmgr
 15  MQSTS-RESOLVEDQMGRNAME PIC X(48).
** Ver:1 **
** Failing object long name
 15  MQSTS-OBJECTSTRING.
** Address of variable length string
 20  MQSTS-OBJECTSTRING-VSPTR POINTER.
** Offset of variable length string
 20  MQSTS-OBJECTSTRING-VSOFFSET PIC S9(9) BINARY.
** Size of buffer
 20  MQSTS-OBJECTSTRING-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
 20  MQSTS-OBJECTSTRING-VSLENGTH PIC S9(9) BINARY.
** CCSID of variable length string
 20  MQSTS-OBJECTSTRING-VSCCSID PIC S9(9) BINARY.
** Failing subscription name
 15  MQSTS-SUBNAME.
** Address of variable length string
 20  MQSTS-SUBNAME-VSPTR POINTER.
** Offset of variable length string
 20  MQSTS-SUBNAME-VSOFFSET PIC S9(9) BINARY.
** Size of buffer
 20  MQSTS-SUBNAME-VSBUFSIZE PIC S9(9) BINARY.
** Length of variable length string
 20  MQSTS-SUBNAME-VSLENGTH PIC S9(9) BINARY.
** CCSID of variable length string
 20  MQSTS-SUBNAME-VSCCSID PIC S9(9) BINARY.
** Failing open options
 15  MQSTS-OPENOPTIONS PIC S9(9) BINARY.
** Failing subscription options
 15  MQSTS-SUBOPTIONS PIC S9(9) BINARY.
** Ver:2 **
```

PL/I declaration:

```
dcl
   1 MQSTS based,
     3 StrucId             char(4),       /* Structure identifier */
      3 Version            fixed bin(31), /* Structure version number */
      3 CompCode           fixed bin(31), /* Completion code */
      3 Reason             fixed bin(31), /* Reason code */
      3 PutSuccessCount    fixed bin(31), /* Put success count */
      3 PutWarningCount    fixed bin(31), /* Put warning count */
      3 PutFailureCount    fixed bin(31), /* Put failure count */
      3 ObjectType         fixed bin(31), /* Object type */
      3 ObjectName         char(48), /* Object name */
      3 ObjectQmgrName     char(48), /* Object queue manager */
      3 ResolvedObjectName char(48), /* Resolved Object name */
      3 ResolvedQmgrName   char(48); /* Resolved Object queue manager */
  /* Ver:1 */
      3 ObjectString,                /* Failing object long name */
         5 VSPtr pointer,            /* Address of variable length string */
         5 VSOffset fixed bin(31),   /* Offset of variable length string */
         5 VSBufSize fixed bin(31),  /* Size of buffer */
         5 VSLength fixed bin(31),   /* Length of variable length string */
         5 VSCCSID fixed bin(31);    /* CCSID of variable length string */
      3 SubName,                     /* Failing subscription name */
         5 VSPtr pointer,            /* Address of variable length string */
         5 VSOffset fixed bin(31),   /* Offset of variable length string */
         5 VSBufSize fixed bin(31),  /* Size of buffer */
         5 VSLength fixed bin(31),   /* Length of variable length string */
         5 VSCCSID fixed bin(31);    /* CCSID of variable length string */
     3 OpenOptions fixed bin(31),    /* Failing open options */
     3 SubOptions fixed bin(31);     /* Failing subscription options */
   /* Ver:2 */
```

High Level Assembler declaration:

```
    MQSTS                          DSECT
    MQSTS_STRUCID                  DS    CL4   Structure identifier
    MQSTS_VERSION                  DS    F     Structure version number
    MQSTS_COMPCODE                 DS    F     Completion code
    MQSTS_REASON                   DS    F     Reason code
    MQSTS_PUTSUCCESSCOUNT          DS    F     Success count
    MQSTS_PUTWARNINGCOUNT          DS    F     Warning count
    MQSTS_PUTFAILURECOUNT          DS    F     Failure count
    MQSTS_OBJTYPE                  DS    F     Object type
    MQSTS_OBJNAME                  DS    CL48  Object name
    MQSTS_OBJQMGR                  DS    CL48  Object queue manager
    MQSTS_ROBJNAME                 DS    CL48  Resolved object name
    MQSTS_ROBJQMGR                 DS    CL48  Resolved object queue manager
    MQSTS_OBJECTSTRING             DS    0F   Force fullword alignment
    MQSTS_OBJECTSTRING_VSPTR       DS    A  Address of variable length string
    MQSTS_OBJECTSTRING_VSOFFSET    DS    F  Offset of variable length string
    MQSTS_OBJECTSTRING_VSBUFSIZE   DS    F  Size of buffer
    MQSTS_OBJECTSTRING_VSLENGTH    DS    F  Length of variable length string
    MQSTS_OBJECTSTRING_VSCCSID     DS    F  CCSID of variable length string
    MQSTS_OBJECTSTRING_LENGTH      EQU   *-MQSTS_OBJECTSTRING
                                         ORG   MQSTS_OBJECTSTRING
    MQSTS_OBJECTSTRING_AREA        DS    CL(MQSTS_OBJECTSTRING_LENGTH)
    *
    MQSTS_SUBNAME                  DS    0F Force fullword alignment
    MQSTS_SUBNAME_VSPTR            DS    A  Address of variable length string
    MQSTS_SUBNAME_VSOFFSET         DS    F  Offset of variable length string
    MQSTS_SUBNAME_VSBUFSIZE        DS    F  Size of buffer
    MQSTS_SUBNAME_VSLENGTH         DS    F  Length of variable length string
    MQSTS_SUBNAME_VSCCSID          DS    F  CCSID of variable length string
    MQSTS_SUBNAME_LENGTH           EQ    *-MQSTS_SUBNAME
                                         ORG   MQSTS_SUBNAME
    MQSTS_SUBNAME_AREA             DS    CL(MQSTS_SUBNAME_LENGTH)
    *
```

```
MQSTS_OPENOPTIONS            DS    F  Failing open options
MQSTS_SUBOPTIONS             DS    F  Failing subscription option
MQSTS_LENGTH                 EQU   *-MQSTS
                             ORG   MQSTS
MQSTS_AREA              DS   CL(MQSTS_LENGTH)
```

MQTM - Trigger message:

The following table summarizes the fields in the structure.

Table 200. Fields in MQTM

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
QName	Name of triggered queue	QName
ProcessName	Name of process object	ProcessName
TriggerData	Trigger data	TriggerData
ApplType	Application type	ApplType
ApplId	Application identifier	ApplId
EnvData	Environment data	EnvData
UserData	User data	UserData

Overview for MQTM:

Purpose: The MQTM structure describes the data in the trigger message that is sent by the queue manager to a trigger-monitor application when a trigger event occurs for a queue.

This structure is part of the WebSphere MQ Trigger Monitor Interface (TMI), which is one of the WebSphere MQ framework interfaces.

Format name: MQFMT_TRIGGER.

Character set and encoding: Character data in MQTM is in the character set of the queue manager that generates the MQTM. Numeric data in MQTM is in the machine encoding of the queue manager that generates the MQTM.

The character set and encoding of the MQTM are given by the *CodedCharSetId* and *Encoding* fields in:

- The MQMD (if the MQTM structure is at the start of the message data), or
- The header structure that precedes the MQTM structure (all other cases).

Usage: A trigger-monitor application might need to pass some or all of the information in the trigger message to the application that the trigger-monitor application starts. Information that might be needed by the started application includes *QName*, *TriggerData*, and *UserData*. The trigger-monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application. For information about MQTMC2, see "MQTMC2 - Trigger message 2 (character format)" on page 1904.

- On z/OS, for an MQAT_CICS application that is started using the CKTI transaction, the entire trigger message structure MQTM is made available to the started transaction; the information can be retrieved by using the EXEC CICS RETRIEVE command.
- On IBM i, the trigger-monitor application provided with WebSphere MQ passes an MQTMC2 structure to the started application.

For information about using triggers, see Starting WebSphere MQ applications using triggers.

MQMD for a trigger message: The fields in the MQMD of a trigger message generated by the queue manager are set as follows:

Field in MQMD	Value used
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	MQMT_DATAGRAM
Expiry	MQEI_UNLIMITED
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
CodedCharSetId	Queue manager's *CodedCharSetId* attribute
Format	MQFMT_TRIGGER
Priority	Initiation queue's *DefPriority* attribute
Persistence	MQPER_NOT_PERSISTENT
MsgId	A unique value
CorrelId	MQCI_NONE
BackoutCount	0
ReplyToQ	Blanks
ReplyToQMgr	Name of queue manager
UserIdentifier	Blanks
AccountingToken	MQACT_NONE
ApplIdentityData	Blanks
PutApplType	MQAT_QMGR, or as appropriate for the message channel agent
PutApplName	First 28 bytes of the queue-manager name
PutDate	Date when trigger message is sent
PutTime	Time when trigger message is sent
ApplOriginData	Blanks

An application that generates a trigger message is recommended to set similar values, except for the following:

- The *Priority* field can be set to MQPRI_PRIORITY_AS_Q_DEF (the queue manager will change this to the default priority for the initiation queue when the message is put).
- The *ReplyToQMgr* field can be set to blanks (the queue manager will change this to the name of the local queue manager when the message it put).
- Set the context fields as appropriate for the application.

Fields for MQTM:

The MQTM structure contains the following fields; the fields are described in **alphabetical order**:

ApplId (MQCHAR256):

This is a character string that identifies the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *ApplId* attribute of the process object identified by the *ProcessName* field; see "Attributes for process definitions" on page 2174 for details of this attribute. The content of this data is of no significance to the queue manager.

The meaning of *ApplId* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ requires *ApplId* to be the name of an executable program. The following notes apply to the environments indicated:

- On z/OS, *ApplId* is:

  - A CICS transaction identifier, for applications started using the CICS trigger-monitor transaction CKTI
  - An IMS transaction identifier, for applications started using the IMS trigger monitor CSQQTRMN
- On Windows systems, the program name can be prefixed with a drive and directory path.
- On IBM i, the program name can be prefixed with a library name and / character.
- On UNIX systems, the program name can be prefixed with a directory path.

The length of this field is given by MQ_PROCESS_APPL_ID_LENGTH. The initial value of this field is the null string in C, and 256 blank characters in other programming languages.

ApplType (MQLONG):

This identifies the nature of the program to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *ApplType* attribute of the process object identified by the *ProcessName* field; see "Attributes for process definitions" on page 2174 for details of this attribute. The content of this data is of no significance to the queue manager.

ApplType can have one of the following standard values. User-defined types can also be used, but should be restricted to values in the range MQAT_USER_FIRST through MQAT_USER_LAST:

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_BATCH
Batch application

MQAT_BROKER
Broker application

MQAT_CICS
CICS transaction.

MQAT_CICS_BRIDGE
CICS bridge application.

MQAT_CICS_VSE
CICS/VSE transaction.

MQAT_DOS
WebSphere MQ MQI client application on PC DOS.

MQAT_IMS
IMS application.

MQAT_IMS_BRIDGE
IMS bridge application.

MQAT_JAVA
Java application.

MQAT_MVS
MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT
Lotus Notes Agent application.

MQAT_NSK
HP Integrity NonStop Server application.

MQAT_OS390
OS/390 application (same value as MQAT_ZOS).

MQAT_OS400
IBM i application.

MQAT_RRS_BATCH
RRS batch application.

MQAT_UNIX
UNIX application.

MQAT_UNKNOWN
Application of unknown type.

MQAT_USER
User-defined application type.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

MQAT_XCF
XCF.

MQAT_ZOS
z/OS application.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

The initial value of this field is `0`.

EnvData (MQCHAR128):

This is a character string that contains environment-related information pertaining to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *EnvData* attribute of the process object identified by the *ProcessName* field; see "Attributes for process definitions" on page 2174 for details of this attribute. The content of this data is of no significance to the queue manager.

On z/OS, for a CICS application started using the CKTI transaction, or an IMS application to be started using the CSQQTRMN transaction, this information is not used.

The length of this field is given by MQ_PROCESS_ENV_DATA_LENGTH. The initial value of this field is the null string in C, and 128 blank characters in other programming languages.

ProcessName (MQCHAR48):

This is the name of the queue-manager process object specified for the triggered queue, and can be used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *ProcessName* attribute of the queue identified by the *QName* field; see "Attributes for queues" on page 2135 for details of this attribute.

Names that are shorter than the defined length of the field are always padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by MQ_PROCESS_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

QName (MQCHAR48):

This is the name of the queue for which a trigger event occurred, and is used by the application started by the trigger-monitor application. The queue manager initializes this field with the value of the *QName* attribute of the triggered queue; see "Attributes for queues" on page 2135 for details of this attribute.

Names that are shorter than the defined length of the field are padded to the right with blanks; they are not ended prematurely by a null character.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQTM_STRUC_ID
Identifier for trigger message structure.

For the C programming language, the constant MQTM_STRUC_ID_ARRAY is also defined; this has the same value as MQTM_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQTM_STRUC_ID.

TriggerData (MQCHAR64):

This is free-format data for use by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *TriggerData* attribute of the queue identified by the *QName* field; see "Attributes for queues" on page 2135 for details of this attribute. The content of this data is of no significance to the queue manager.

On z/OS, for a CICS application started using the CKTI transaction, this information is not used.

The length of this field is given by MQ_TRIGGER_DATA_LENGTH. The initial value of this field is the null string in C, and 64 blank characters in other programming languages.

UserData (MQCHAR128):

This is a character string that contains user information relevant to the application to be started, and is used by the trigger-monitor application that receives the trigger message. The queue manager initializes this field with the value of the *UserData* attribute of the process object identified by the *ProcessName* field; see "Attributes for process definitions" on page 2174 for details of this attribute. The content of this data is of no significance to the queue manager.

For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

The length of this field is given by MQ_PROCESS_USER_DATA_LENGTH. The initial value of this field is the null string in C, and 128 blank characters in other programming languages.

Version (MQLONG):

This is the structure version number. The value must be:

MQTM_VERSION_1
Version number for trigger message structure.

The following constant specifies the version number of the current version:

MQTM_CURRENT_VERSION
Current version of trigger message structure.

The initial value of this field is MQTM_VERSION_1.

Initial values and language declarations for MQTM:

Table 201. Initial values of fields in MQTM for MQTM

Field name	Name of constant	Value of constant
StrucId	MQTM_STRUC_ID	'TM¬¬'
Version	MQTM_VERSION_1	1
QName	None	Null string or blanks
ProcessName	None	Null string or blanks
TriggerData	None	Null string or blanks
ApplType	None	0
ApplId	None	Null string or blanks
EnvData	None	Null string or blanks
UserData	None	Null string or blanks

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQTM_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQTM MyTM = {MQTM_DEFAULT};
```

C declaration:

```
typedef struct tagMQTM MQTM;
struct tagMQTM {
  MQCHAR4    StrucId;      /* Structure identifier */
  MQLONG     Version;      /* Structure version number */
  MQCHAR48   QName;        /* Name of triggered queue */
  MQCHAR48   ProcessName;  /* Name of process object */
  MQCHAR64   TriggerData;  /* Trigger data */
  MQLONG     ApplType;     /* Application type */
  MQCHAR256  ApplId;       /* Application identifier */
  MQCHAR128  EnvData;      /* Environment data */
  MQCHAR128  UserData;     /* User data */
};
```

COBOL declaration:

```
**   MQTM structure
  10 MQTM.
**    Structure identifier
   15 MQTM-STRUCID     PIC X(4).
**    Structure version number
   15 MQTM-VERSION     PIC S9(9) BINARY.
**    Name of triggered queue
   15 MQTM-QNAME       PIC X(48).
**    Name of process object
   15 MQTM-PROCESSNAME PIC X(48).
**    Trigger data
   15 MQTM-TRIGGERDATA PIC X(64).
**    Application type
   15 MQTM-APPLTYPE    PIC S9(9) BINARY.
**    Application identifier
   15 MQTM-APPLID      PIC X(256).
**    Environment data
   15 MQTM-ENVDATA     PIC X(128).
**    User data
   15 MQTM-USERDATA    PIC X(128).
```

PL/I declaration:

```
dcl
 1 MQTM based,
  3 StrucId     char(4),       /* Structure identifier */
  3 Version     fixed bin(31), /* Structure version number */
  3 QName       char(48),      /* Name of triggered queue */
  3 ProcessName char(48),      /* Name of process object */
  3 TriggerData char(64),      /* Trigger data */
  3 ApplType    fixed bin(31), /* Application type */
  3 ApplId      char(256),     /* Application identifier */
  3 EnvData     char(128),     /* Environment data */
  3 UserData    char(128);     /* User data */
```

High Level Assembler declaration:

```
MQTM              DSECT
MQTM_STRUCID      DS   CL4    Structure identifier
MQTM_VERSION      DS   F      Structure version number
MQTM_QNAME        DS   CL48   Name of triggered queue
MQTM_PROCESSNAME  DS   CL48   Name of process object
MQTM_TRIGGERDATA  DS   CL64   Trigger data
MQTM_APPLTYPE     DS   F      Application type
MQTM_APPLID       DS   CL256  Application identifier
MQTM_ENVDATA      DS   CL128  Environment data
MQTM_USERDATA     DS   CL128  User data
*
MQTM_LENGTH       EQU  *-MQTM
                  ORG  MQTM
MQTM_AREA         DS   CL(MQTM_LENGTH)
```

Visual Basic declaration:

```
Type MQTM
  StrucId     As String*4   'Structure identifier'
  Version     As Long       'Structure version number'
  QName       As String*48  'Name of triggered queue'
  ProcessName As String*48  'Name of process object'
  TriggerData As String*64  'Trigger data'
  ApplType    As Long       'Application type'
  ApplId      As String*256 'Application identifier'
  EnvData     As String*128 'Environment data'
  UserData    As String*128 'User data'
End Type
```

MQTMC2 - Trigger message 2 (character format):

The following table summarizes the fields in the structure.

Table 202. Fields in MQTMC2

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
QName	Name of triggered queue	QName
ProcessName	Name of process object	ProcessName
TriggerData	Trigger data	TriggerData
ApplType	Application type	ApplType
ApplId	Application identifier	ApplId
EnvData	Environment data	EnvData
UserData	User data	UserData
QMgrName	Queue manager name	QMgrName

Overview for MQTMC2:

Purpose: When a trigger-monitor application retrieves a trigger message (MQTM) from an initiation queue, the trigger monitor might need to pass some or all of the information in the trigger message to the application that the trigger monitor starts.

Information that the started application might need includes *QName*, *TriggerData*, and *UserData*. The trigger monitor application can pass the MQTM structure directly to the started application, or pass an MQTMC2 structure instead, depending on what is permitted by the environment and convenient for the started application.

This structure is part of the WebSphere MQ Trigger Monitor Interface (TMI), which is one of the WebSphere MQ framework interfaces.

Character set and encoding: Character data in MQTMC2 is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute.

Usage: The MQTMC2 structure is very similar to the format of the MQTM structure. The difference is that the non-character fields in MQTM are changed in MQTMC2 to character fields of the same length, and the queue manager name is added at the end of the structure.

- On z/OS, for an MQAT_IMS application that is started using the CSQQTRMN application, an MQTMC2 structure is made available to the started application.

- On IBM i, the trigger monitor application provided with WebSphere MQ passes an MQTMC2 structure to the started application.

Fields for MQTMC2:

The MQTMC2 structure contains the following fields; the fields are described in **alphabetical order**:

ApplId (MQCHAR256):

Application identifier.

See the *ApplId* field in the MQTM structure.

ApplType (MQCHAR4):

Application type.

This field always contains blanks, whatever the value in the *ApplType* field in the MQTM structure of the original trigger message.

EnvData (MQCHAR128):

Environment data.

See the *EnvData* field in the MQTM structure.

ProcessName (MQCHAR48):

Name of process object.

See the *ProcessName* field in the MQTM structure.

QMgrName (MQCHAR48):

Queue manager name.

This is the name of the queue manager at which the trigger event occurred.

QName (MQCHAR48):

Name of triggered queue.

See the *QName* field in the MQTM structure.

StrucId (MQCHAR4):

Structure identifier.

The value must be:

MQTMC_STRUC_ID
: Identifier for trigger message (character format) structure.

  For the C programming language, the constant MQTMC_STRUC_ID_ARRAY is also defined; this has the same value as MQTMC_STRUC_ID, but is an array of characters instead of a string.

TriggerData (MQCHAR64):

Trigger data.

See the *TriggerData* field in the MQTM structure.

UserData (MQCHAR128):

User data.

See the *UserData* field in the MQTM structure.

Version (MQCHAR4):

Structure version number.

The value must be:

MQTMC_VERSION_2
Version 2 trigger message (character format) structure.

For the C programming language, the constant MQTMC_VERSION_2_ARRAY is also defined; this has the same value as MQTMC_VERSION_2, but is an array of characters instead of a string.

The following constant specifies the version number of the current version:

MQTMC_CURRENT_VERSION
Current version of trigger message (character format) structure.

Initial values and language declarations for MQTMC2:

Table 203. Initial values of fields in MQTMC2 for MQTMC2

Field name	Name of constant	Value of constant
StrucId	MQTMC_STRUC_ID	'TMC⌂'
Version	MQTMC_VERSION_2	'⌂⌂2'
QName	None	Null string or blanks
ProcessName	None	Null string or blanks
TriggerData	None	Null string or blanks
ApplType	None	Blanks
ApplId	None	Null string or blanks
EnvData	None	Null string or blanks
UserData	None	Null string or blanks
QMgrName	None	Null string or blanks

Notes:

1. The symbol ⌂ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQTMC2_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQTMC2 MyTMC = {MQTMC2_DEFAULT};
```

C declaration:

```
typedef struct tagMQTMC2 MQTMC2;
struct tagMQTMC2 {
  MQCHAR4    StrucId;      /* Structure identifier */
  MQCHAR4    Version;      /* Structure version number */
  MQCHAR48   QName;        /* Name of triggered queue */
  MQCHAR48   ProcessName;  /* Name of process object */
  MQCHAR64   TriggerData;  /* Trigger data */
  MQCHAR4    ApplType;     /* Application type */
  MQCHAR256  ApplId;       /* Application identifier */
  MQCHAR128  EnvData;      /* Environment data */
  MQCHAR128  UserData;     /* User data */
  MQCHAR48   QMgrName;     /* Queue manager name */
};
```

COBOL declaration:

```
**   MQTMC2 structure
  10 MQTMC2.
**    Structure identifier
   15 MQTMC2-STRUCID     PIC X(4).
**    Structure version number
   15 MQTMC2-VERSION     PIC X(4).
**    Name of triggered queue
   15 MQTMC2-QNAME       PIC X(48).
**    Name of process object
   15 MQTMC2-PROCESSNAME PIC X(48).
**    Trigger data
   15 MQTMC2-TRIGGERDATA PIC X(64).
**    Application type
   15 MQTMC2-APPLTYPE    PIC X(4).
**    Application identifier
   15 MQTMC2-APPLID      PIC X(256).
**    Environment data
   15 MQTMC2-ENVDATA     PIC X(128).
**    User data
   15 MQTMC2-USERDATA    PIC X(128).
**    Queue manager name
   15 MQTMC2-QMGRNAME    PIC X(48).
```

PL/I declaration:

```
dcl
 1 MQTMC2 based,
  3 StrucId     char(4),   /* Structure identifier */
  3 Version     char(4),   /* Structure version number */
  3 QName       char(48),  /* Name of triggered queue */
  3 ProcessName char(48),  /* Name of process object */
  3 TriggerData char(64),  /* Trigger data */
  3 ApplType    char(4),   /* Application type */
  3 ApplId      char(256), /* Application identifier */
  3 EnvData     char(128), /* Environment data */
  3 UserData    char(128), /* User data */
  3 QMgrName    char(48);  /* Queue manager name */
```

High Level Assembler declaration:

```
MQTMC             DSECT
MQTMC_STRUCID     DS   CL4    Structure identifier
MQTMC_VERSION     DS   CL4    Structure version number
MQTMC_QNAME       DS   CL48   Name of triggered queue
MQTMC_PROCESSNAME DS   CL48   Name of process object
MQTMC_TRIGGERDATA DS   CL64   Trigger data
MQTMC_APPLTYPE    DS   CL4    Application type
MQTMC_APPLID      DS   CL256  Application identifier
MQTMC_ENVDATA     DS   CL128  Environment data
MQTMC_USERDATA    DS   CL128  User data
MQTMC_QMGRNAME    DS   CL48   Queue manager name
*
MQTMC_LENGTH      EQU  *-MQTMC
                  ORG  MQTMC
MQTMC_AREA        DS   CL(MQTMC_LENGTH)
```

Visual Basic declaration:

```
Type MQTMC2
  StrucId     As String*4   'Structure identifier'
  Version     As String*4   'Structure version number'
  QName       As String*48  'Name of triggered queue'
  ProcessName As String*48  'Name of process object'
  TriggerData As String*64  'Trigger data'
  ApplType    As String*4   'Application type'
  ApplId      As String*256 'Application identifier'
  EnvData     As String*128 'Environment data'
  UserData    As String*128 'User data'
  QMgrName    As String*48  'Queue manager name'
End Type
```

MQWIH - Work information header:

The following table summarizes the fields in the structure.

Table 204. Fields in MQWIH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
StrucLength	Length of MQWIH structure	StrucLength
Encoding	Numeric encoding of data that follows MQWIH	Encoding
CodedCharSetId	Character-set identifier of data that follows MQWIH	CodedCharSetId
Format	Format name of data that follows MQWIH	Format
Flags	Flags	Flags
ServiceName	Service name	ServiceName
ServiceStep	Service step name	ServiceStep
MsgToken	Message token	MsgToken
Reserved	Reserved	Reserved

Overview for MQWIH:

Availability: All WebSphere MQ systems, plus WebSphere MQ clients connected to these systems.

Purpose: The MQWIH structure describes the information that must be present at the start of a message that is to be handled by the z/OS workload manager.

Format name: MQFMT_WORK_INFO_HEADER.

Character set and encoding: The fields in the MQWIH structure are in the character set and encoding given by the *CodedCharSetId* and *Encoding* fields in the header structure that precedes MQWIH, or by those fields in the MQMD structure if the MQWIH is at the start of the application message data.

The character set must be one that has single-byte characters for the characters that are valid in queue names.

Usage: If a message is to be processed by the z/OS workload manager, the message must begin with an MQWIH structure.

Fields for MQWIH:

The MQWIH structure contains the following fields; the fields are described in **alphabetical order**:

CodedCharSetId (MQLONG):

This specifies the character set identifier of the data that follows the MQWIH structure; it does not apply to character data in the MQWIH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. You can use the following special value:

MQCCSI_INHERIT
Character data in the data *following* this structure is in the same character set as this structure.

The queue manager changes this value in the structure sent in the message to the actual character-set identifier of the structure. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_INHERIT cannot be used if the value of the *PutApplType* field in MQMD is MQAT_BROKER.

The initial value of this field is MQCCSI_UNDEFINED.

Encoding (MQLONG):

This specifies the numeric encoding of the data that follows the MQWIH structure; it does not apply to numeric data in the MQWIH structure itself.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data.

The initial value of this field is 0.

Flags (MQLONG):

The value must be:

MQWIH_NONE
No flags.

The initial value of this field is MQWIH_NONE.

Format (MQCHAR8):

This specifies the format name of the data that follows the MQWIH structure.

On the MQPUT or MQPUT1 call, the application must set this field to the value appropriate to the data. The rules for coding this field are the same as those for the *Format* field in MQMD.

The length of this field is given by MQ_FORMAT_LENGTH. The initial value of this field is MQFMT_NONE.

MsgToken (MQBYTE16):

This is a message token that uniquely identifies the message.

For the MQPUT and MQPUT1 calls, this field is ignored. The length of this field is given by MQ_MSG_TOKEN_LENGTH. The initial value of this field is MQMTOK_NONE.

Reserved (MQCHAR32):

This is a reserved field; it must be blank.

ServiceName (MQCHAR32):

This is the name of the service that is to process the message.

The length of this field is given by MQ_SERVICE_NAME_LENGTH. The initial value of this field is 32 blank characters.

ServiceStep (MQCHAR8):

This is the name of the step of *ServiceName* to which the message relates.

The length of this field is given by MQ_SERVICE_STEP_LENGTH. The initial value of this field is 8 blank characters.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQWIH_STRUC_ID
Identifier for work information header structure.

For the C programming language, the constant MQWIH_STRUC_ID_ARRAY is also defined; this has the same value as MQWIH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQWIH_STRUC_ID.

StrucLength (MQLONG):

This is the length of the MQWIH structure. The value must be:

MQWIH_LENGTH_1
Length of version-1 work information header structure.

The following constant specifies the length of the current version:

MQWIH_CURRENT_LENGTH
Length of current version of work information header structure.

The initial value of this field is MQWIH_LENGTH_1.

Version (MQLONG):

This is the structure version number. The value must be:

MQWIH_VERSION_1
Version-1 work information header structure.

The following constant specifies the version number of the current version:

MQWIH_CURRENT_VERSION
Current version of work information header structure.

The initial value of this field is MQWIH_VERSION_1.

Initial values and language declarations for MQWIH:

Table 205. Initial values of fields in MQWIH for MQWIH

Field name	Name of constant	Value of constant
StrucId	MQWIH_STRUC_ID	'WIH¬'
Version	MQWIH_VERSION_1	1
StrucLength	MQWIH_LENGTH_1	120
Encoding	None	0
CodedCharSetId	MQCCSI_UNDEFINED	0
Format	MQFMT_NONE	Blanks
Flags	MQWIH_NONE	0
ServiceName	None	Blanks
ServiceStep	None	Blanks
MsgToken	MQMTOK_NONE	Nulls
Reserved	None	Blanks

Notes:

1. The symbol ¬ represents a single blank character.
2. In the C programming language, the macro variable MQWIH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQWIH MyWIH = {MQWIH_DEFAULT};
```

C declaration:

```
typedef struct tagMQWIH MQWIH;
struct tagMQWIH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQLONG    StrucLength;     /* Length of MQWIH structure */
  MQLONG    Encoding;        /* Numeric encoding of data that follows
                                MQWIH */
  MQLONG    CodedCharSetId;  /* Character-set identifier of data that
                                follows MQWIH */
  MQCHAR8   Format;          /* Format name of data that follows
                                MQWIH */
  MQLONG    Flags;           /* Flags */
  MQCHAR32  ServiceName;     /* Service name */
  MQCHAR8   ServiceStep;     /* Service step name */
  MQBYTE16  MsgToken;        /* Message token */
  MQCHAR32  Reserved;        /* Reserved */
};
```

COBOL declaration:

```
**   MQWIH structure
  10 MQWIH.
**    Structure identifier
   15 MQWIH-STRUCID        PIC X(4).
**    Structure version number
   15 MQWIH-VERSION        PIC S9(9) BINARY.
**    Length of MQWIH structure
   15 MQWIH-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding of data that follows MQWIH
   15 MQWIH-ENCODING       PIC S9(9) BINARY.
**    Character-set identifier of data that follows MQWIH
   15 MQWIH-CODEDCHARSETID PIC S9(9) BINARY.
**    Format name of data that follows MQWIH
   15 MQWIH-FORMAT         PIC X(8).
**    Flags
   15 MQWIH-FLAGS          PIC S9(9) BINARY.
**    Service name
   15 MQWIH-SERVICENAME    PIC X(32).
**    Service step name
   15 MQWIH-SERVICESTEP    PIC X(8).
**    Message token
   15 MQWIH-MSGTOKEN       PIC X(16).
**    Reserved
   15 MQWIH-RESERVED       PIC X(32).
```

PL/I declaration:

```
dcl
 1 MQWIH based,
  3 StrucId        char(4),       /* Structure identifier */
  3 Version        fixed bin(31), /* Structure version number */
  3 StrucLength    fixed bin(31), /* Length of MQWIH structure */
  3 Encoding       fixed bin(31), /* Numeric encoding of data that
                                     follows MQWIH */
  3 CodedCharSetId fixed bin(31), /* Character-set identifier of data
                                     that follows MQWIH */
  3 Format         char(8),       /* Format name of data that follows
                                     MQWIH */
  3 Flags          fixed bin(31), /* Flags */
  3 ServiceName    char(32),      /* Service name */
  3 ServiceStep    char(8),       /* Service step name */
  3 MsgToken       char(16),      /* Message token */
  3 Reserved       char(32);      /* Reserved */
```

High Level Assembler declaration:

```
MQWIH                 DSECT
MQWIH_STRUCID         DS   CL4   Structure identifier
MQWIH_VERSION         DS   F     Structure version number
MQWIH_STRUCLENGTH     DS   F     Length of MQWIH structure
MQWIH_ENCODING        DS   F     Numeric encoding of data that follows
*                                MQWIH
MQWIH_CODEDCHARSETID  DS   F     Character-set identifier of data that
*                                follows MQWIH
MQWIH_FORMAT          DS   CL8   Format name of data that follows MQWIH
MQWIH_FLAGS           DS   F     Flags
MQWIH_SERVICENAME     DS   CL32  Service name
MQWIH_SERVICESTEP     DS   CL8   Service step name
MQWIH_MSGTOKEN        DS   XL16  Message token
MQWIH_RESERVED        DS   CL32  Reserved
*
MQWIH_LENGTH          EQU  *-MQWIH
                      ORG  MQWIH
MQWIH_AREA            DS   CL(MQWIH_LENGTH)
```

Visual Basic declaration:

```
Type MQWIH
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  StrucLength    As Long      'Length of MQWIH structure'
  Encoding       As Long      'Numeric encoding of data that follows'
                              'MQWIH'
  CodedCharSetId As Long      'Character-set identifier of data that'
                              'follows MQWIH'
  Format         As String*8  'Format name of data that follows MQWIH'
  Flags          As Long      'Flags'
  ServiceName    As String*32 'Service name'
  ServiceStep    As String*8  'Service step name'
  MsgToken       As MQBYTE16  'Message token'
  Reserved       As String*32 'Reserved'
End Type
```

MQXP - Exit parameter block:

The following table summarizes the fields in the structure.

Table 206. Fields in MQXP

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
ExitId	Exit identifier	ExitId
ExitReason	Reason for invocation of exit	ExitReason
ExitResponse	Response from exit	ExitResponse
ExitCommand	API call code	ExitCommand
ExitParmCount	Parameter count	ExitParmCount
ExitUserArea	User area	ExitUserArea

Overview for MQXP:

Availability: z/OS.

Purpose: The MQXP structure is used as an input/output parameter to the API-crossing exit. For more information about this exit, see The API-crossing exit.

Character set and encoding: Character data in MQXP is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQXP is in the native machine encoding; this is given by MQENC_NATIVE.

Fields for MQXP:

The MQXP structure contains the following fields; the fields are described in **alphabetical order**:

ExitCommand (MQLONG):

This field is set on entry to the exit routine. It identifies the API call that caused the exit to be invoked:

MQXC_CALLBACK
The CALLBACK call.

MQXC_MQBACK
The MQBACK call.

MQXC_MQCB
The MQCB call.

MQXC_MQCLOSE
The MQCLOSE call.

MQXC_MQCMIT
The MQCMIT call.

MQXC_MQCTL
The MQCTL call.

MQXC_MQGET
The MQGET call.

MQXC_MQINQ
The MQINQ call.

MQXC_MQOPEN
The MQOPEN call.

MQXC_MQPUT
The MQPUT call.

MQXC_MQPUT1
The MQPUT1 call.

MQXC_MQSET
The MQSET call.

MQXC_MQSTAT
The MQSTAT call.

MQXC_MQSUB
The MQSUB call.

MQXC_MQSUBRQ
The MQSUBRQ call.

This is an input field to the exit.

ExitId (MQLONG):

This is set on entry to the exit routine, and indicates the type of exit:

MQXT_API_CROSSING_EXIT
API-crossing exit for CICS.

This is an input field to the exit.

ExitParmCount (MQLONG):

This field is set on entry to the exit routine. It contains the number of parameters that the MQ call takes. These are:

Call name	Number of parameters
MQBACK	3
MQCLOSE	5
MQCMIT	3
MQGET	9
MQINQ	10
MQOPEN	6
MQPUT	8
MQPUT1	8
MQSET	10

This is an input field to the exit.

ExitReason (MQLONG):

This is set on entry to the exit routine. For the API-crossing exit it indicates whether the routine is called before or after execution of the API call:

MQXR_BEFORE
Before API execution.

MQXR_AFTER
After API execution.

This is an input field to the exit.

ExitResponse (MQLONG):

The value is set by the exit to communicate with the caller. The following values are defined:

MQXCC_OK
Exit completed successfully.

MQXCC_SUPPRESS_FUNCTION
Suppress function.

When this value is set by an API-crossing exit called *before* the API call, the API call is not performed. The *CompCode* for the call is set to MQCC_FAILED, the *Reason* is set to MQRC_SUPPRESSED_BY_EXIT, and all other parameters remain as the exit left them.

When this value is set by an API-crossing exit called *after* the API call, it is ignored by the queue manager.

MQXCC_SKIP_FUNCTION
Skip function.

When this value is set by an API-crossing exit called *before* the API call, the API call is not performed; the *CompCode* and *Reason* and all other parameters remain as the exit left them.

When this value is set by an API-crossing exit called *after* the API call, it is ignored by the queue manager.

This is an output field from the exit.

ExitUserArea (MQBYTE16):

This is a field that is available for the exit to use. It is initialized to binary zero for the length of the field before the first invocation of the exit for the task, and thereafter any changes made to this field by the exit are preserved across invocations of the exit. The following value is defined:

MQXUA_NONE
No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

Reserved (MQLONG):

This is a reserved field. Its value is not significant to the exit.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQXP_STRUC_ID
Identifier for exit parameter structure.

For the C programming language, the constant MQXP_STRUC_ID_ARRAY is also defined; this has the same value as MQXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG):

This is the structure version number. The value must be:

MQXP_VERSION_1
Version number for exit parameter-block structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit must therefore check that the version number is equal to or greater than the lowest version that contains the fields that the exit needs to use.

This is an input field to the exit.

Language declarations:

This structure is supported in the following programming languages.

C declaration:

```
typedef struct tagMQXP MQXP;
struct tagMQXP {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQLONG    ExitId;         /* Exit identifier */
  MQLONG    ExitReason;     /* Reason for invocation of exit */
  MQLONG    ExitResponse;   /* Response from exit */
  MQLONG    ExitCommand;    /* API call code */
  MQLONG    ExitParmCount;  /* Parameter count */
  MQLONG    Reserved;       /* Reserved */
  MQBYTE16  ExitUserArea;   /* User area */
};
```

COBOL declaration:

```
**   MQXP structure
  10 MQXP.
**    Structure identifier
   15 MQXP-STRUCID       PIC X(4).
**    Structure version number
   15 MQXP-VERSION       PIC S9(9) BINARY.
**    Exit identifier
   15 MQXP-EXITID        PIC S9(9) BINARY.
**    Reason for invocation of exit
   15 MQXP-EXITREASON    PIC S9(9) BINARY.
**    Response from exit
   15 MQXP-EXITRESPONSE  PIC S9(9) BINARY.
**    API call code
   15 MQXP-EXITCOMMAND   PIC S9(9) BINARY.
**    Parameter count
   15 MQXP-EXITPARMCOUNT PIC S9(9) BINARY.
**    Reserved
   15 MQXP-RESERVED      PIC S9(9) BINARY.
**    User area
   15 MQXP-EXITUSERAREA  PIC X(16).
```

PL/I declaration:

```
dcl
 1 MQXP based,
  3 StrucId       char(4),       /* Structure identifier */
  3 Version       fixed bin(31), /* Structure version number */
  3 ExitId        fixed bin(31), /* Exit identifier */
  3 ExitReason    fixed bin(31), /* Reason for invocation of exit */
  3 ExitResponse  fixed bin(31), /* Response from exit */
  3 ExitCommand   fixed bin(31), /* API call code */
  3 ExitParmCount fixed bin(31), /* Parameter count */
  3 Reserved      fixed bin(31), /* Reserved */
  3 ExitUserArea  char(16);      /* User area */
```

High Level Assembler declaration:

```
MQXP                DSECT
MQXP_STRUCID        DS   CL4   Structure identifier
MQXP_VERSION        DS   F     Structure version number
MQXP_EXITID         DS   F     Exit identifier
MQXP_EXITREASON     DS   F     Reason for invocation of exit
MQXP_EXITRESPONSE   DS   F     Response from exit
MQXP_EXITCOMMAND    DS   F     API call code
MQXP_EXITPARMCOUNT  DS   F     Parameter count
MQXP_RESERVED       DS   F     Reserved
MQXP_EXITUSERAREA   DS   XL16  User area
*
MQXP_LENGTH         EQU  *-MQXP
                    ORG  MQXP
MQXP_AREA           DS   CL(MQXP_LENGTH)
```

MQXQH - Transmission-queue header:

The following table summarizes the fields in the structure.

Table 207. Fields in MQXQH

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
RemoteQName	Name of destination queue	RemoteQName
RemoteQMgrName	Name of destination queue manager	RemoteQMgrName
MsgDesc	Original message descriptor	MsgDesc

Overview for MQXQH:

Availability: All WebSphere MQ systems and WebSphere MQ clients.

Purpose: The MQXQH structure describes the information that is prefixed to the application message data of messages when they are on transmission queues. A transmission queue is a special type of local queue that temporarily holds messages destined for remote queues (that is, destined for queues that do not belong to the local queue manager). A transmission queue is denoted by the *Usage* queue attribute having the value MQUS_TRANSMISSION.

Format name: MQFMT_XMIT_Q_HEADER.

Character set and encoding: Data in MQXOH must be in the character set given by the *CodedCharSetId* queue-manager attribute and encoding of the local queue manager given by MQENC_NATIVE.

Set the character set and encoding of the MQXQH into the *CodedCharSetId* and *Encoding* fields in:

- The separate MQMD (if the MQXQH structure is at the start of the message data), or
- The header structure that precedes the MQXQH structure (all other cases).

Usage: A message that is on a transmission queue has *two* message descriptors:

- One message descriptor is stored separately from the message data; this is called the *separate message descriptor*, and is generated by the queue manager when the message is placed on the transmission queue. Some of the fields in the separate message descriptor are copied from the message descriptor provided by the application on the MQPUT or MQPUT1 call.

  The separate message descriptor is the one that is returned to the application in the *MsgDesc* parameter of the MQGET call when the message is removed from the transmission queue.

- A second message descriptor is stored within the MQXQH structure as part of the message data; this is called the *embedded message descriptor*, and is a copy of the message descriptor that was provided by the application on the MQPUT or MQPUT1 call (with minor variations).

  The embedded message descriptor is always a version-1 MQMD. If the message put by the application has nondefault values for one or more of the version-2 fields in the MQMD, an MQMDE structure follows the MQXQH, and is in turn followed by the application message data (if any). The MQMDE is either:

  – Generated by the queue manager (if the application uses a version-2 MQMD to put the message), or
  – Already present at the start of the application message data (if the application uses a version-1 MQMD to put the message).

  The embedded message descriptor is the one that is returned to the application in the *MsgDesc* parameter of the MQGET call when the message is removed from the final destination queue.

Fields in the separate message descriptor: The fields in the separate message descriptor are set by the queue manager as shown. If the queue manager does not support the version-2 MQMD, a version-1 MQMD is used without loss of function.

Field in separate MQMD	Value used
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_2
Report	Copied from the embedded message descriptor, but with the bits identified by MQRO_ACCEPT_UNSUP_IF_XMIT_MASK set to zero. (This prevents a COA or COD report message being generated when a message is placed on or removed from a transmission queue.)
MsgType	Copied from the embedded message descriptor.
Expiry	Copied from the embedded message descriptor.
Feedback	Copied from the embedded message descriptor.
Encoding	MQENC_NATIVE (see note)
CodedCharSetId	Queue manager's *CodedCharSetId* attribute.
Format	MQFMT_XMIT_Q_HEADER
Priority	Copied from the embedded message descriptor.
Persistence	Copied from the embedded message descriptor.
MsgId	A new value is generated by the queue manager. This message identifier is different from the *MsgId* that the queue manager may have generated for the embedded message descriptor (see above).
CorrelId	The *MsgId* from the embedded message descriptor. For messages being put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE, *CorrelId* is reserved for internal use.
BackoutCount	0
ReplyToQ	Copied from the embedded message descriptor.
ReplyToQMgr	Copied from the embedded message descriptor.
UserIdentifier	Copied from the embedded message descriptor.
AccountingToken	Copied from the embedded message descriptor. For messages being put to the SYSTEM.CLUSTER.TRANSMIT.QUEUE, *AccountingToken* is reserved for internal use.
ApplIdentityData	Copied from the embedded message descriptor.
PutApplType	MQAT_QMGR
PutApplName	First 28 bytes of the queue-manager name.
PutDate	Date when message was put on transmission queue.
PutTime	Time when message was put on transmission queue.
ApplOriginData	Blanks
GroupId	MQGI_NONE
MsgSeqNumber	1
Offset	0
MsgFlags	MQMF_NONE
OriginalLength	MQOL_UNDEFINED

- On Windows, the value of MQENC_NATIVE for Micro Focus COBOL differs from the value for C. The value in the *Encoding* field in the separate message descriptor is always the value for C in these environments; this value is 546 in decimal. Also, the integer fields in the MQXQH structure are in the encoding that corresponds to this value (the native Intel encoding).

Fields in the embedded message descriptor: The fields in the embedded message descriptor have the same values as those in the *MsgDesc* parameter of the MQPUT or MQPUT1 call, except for the following:

- The *Version* field always has the value MQMD_VERSION_1.
- If the *Priority* field has the value MQPRI_PRIORITY_AS_Q_DEF, it is replaced by the value of the queue's *DefPriority* attribute.
- If the *Persistence* field has the value MQPER_PERSISTENCE_AS_Q_DEF, it is replaced by the value of the queue's *DefPersistence* attribute.
- If the *MsgId* field has the value MQMI_NONE, or the MQPMO_NEW_MSG_ID option was specified, or the message is a distribution-list message, *MsgId* is replaced by a new message identifier generated by the queue manager.

  When a distribution-list message is split into smaller distribution-list messages placed on different transmission queues, the *MsgId* field in each of the new embedded message descriptors is the same as that in the original distribution-list message.
- If the MQPMO_NEW_CORREL_ID option was specified, *CorrelId* is replaced by a new correlation identifier generated by the queue manager.
- The context fields are set as indicated by the MQPMO_*_CONTEXT options specified in the *PutMsgOpts* parameter; the context fields are:
  - *AccountingToken*
  - *ApplIdentityData*
  - *ApplOriginData*
  - *PutApplName*
  - *PutApplType*
  - *PutDate*
  - *PutTime*
  - *UserIdentifier*
- The version-2 fields (if they were present) are removed from the MQMD, and moved into an MQMDE structure, if one or more of the version-2 fields has a nondefault value.

Putting messages on remote queues: When an application puts a message on a remote queue (either by specifying the name of the remote queue directly, or by using a local definition of the remote queue), the local queue manager:

- Creates an MQXQH structure containing the embedded message descriptor
- Appends an MQMDE if one is needed and is not already present
- Appends the application message data
- Places the message on an appropriate transmission queue

Putting messages directly on transmission queues: An application can also put a message directly on a transmission queue. In this case the application must prefix the application message data with an MQXQH structure, and initialize the fields with appropriate values. In addition, the *Format* field in the *MsgDesc* parameter of the MQPUT or MQPUT1 call must have the value MQFMT_XMIT_Q_HEADER.

Character data in the MQXQH structure created by the application must be in the character set of the local queue manager (defined by the *CodedCharSetId* queue-manager attribute), and integer data must be in the native machine encoding. In addition, character data in the MQXQH structure must be padded

with blanks to the defined length of the field; the data must not be ended prematurely by using a null character, because the queue manager does not convert the null and subsequent characters to blanks in the MQXQH structure.

However, the queue manager does not check that an MQXQH structure is present, or that valid values have been specified for the fields.

Applications should not put their messages directly to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Getting messages from transmission queues: Applications that get messages from a transmission queue must process the information in the MQXQH structure in an appropriate fashion. The presence of the MQXQH structure at the beginning of the application message data is indicated by the value MQFMT_XMIT_Q_HEADER being returned in the *Format* field in the *MsgDesc* parameter of the MQGET call. The values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter indicate the character set and encoding of the character and integer data in the MQXQH structure. The character set and encoding of the application message data are defined by the *CodedCharSetId* and *Encoding* fields in the embedded message descriptor.

Fields for MQXQH:

The MQXQH structure contains the following fields; the fields are described in **alphabetical order**:

MsgDesc (MQMD1):

This is the embedded message descriptor, and is a close copy of the message descriptor MQMD that was specified as the *MsgDesc* parameter on the MQPUT or MQPUT1 call when the message was originally put to the remote queue.

Note: This is a version-1 MQMD.

The initial values of the fields in this structure are the same as those in the MQMD structure.

RemoteQMgrName (MQCHAR48):

This is the name of the queue manager or queue-sharing group that owns the queue that is the apparent eventual destination for the message.

If the message is a distribution-list message, *RemoteQMgrName* is blank.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

RemoteQName (MQCHAR48):

This is the name of the message queue that is the apparent eventual destination for the message (this might prove not to be the eventual destination if, for example, this queue is defined at *RemoteQMgrName* to be a local definition of another remote queue).

If the message is a distribution-list message (that is, the *Format* field in the embedded message descriptor is MQFMT_DIST_HEADER), *RemoteQName* is blank.

The length of this field is given by MQ_Q_NAME_LENGTH. The initial value of this field is the null string in C, and 48 blank characters in other programming languages.

StrucId (MQCHAR4):

This is the structure identifier. The value must be:

MQXQH_STRUC_ID
Identifier for transmission-queue header structure.

For the C programming language, the constant MQXQH_STRUC_ID_ARRAY is also defined; this has the same value as MQXQH_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQXQH_STRUC_ID.

Version (MQLONG):

This is the structure version number. The value must be:

MQXQH_VERSION_1
Version number for transmission-queue header structure.

The following constant specifies the version number of the current version:

MQXQH_CURRENT_VERSION
Current version of transmission-queue header structure.

The initial value of this field is MQXQH_VERSION_1.

Initial values and language declarations for MQXQH:

Table 208. Initial values of fields in MQXQH for MQXQH

Field name	Name of constant	Value of constant
StrucId	MQXQH_STRUC_ID	'XQH¬'
Version	MQXQH_VERSION_1	1
RemoteQName	None	Null string or blanks
RemoteQMgrName	None	Null string or blanks
MsgDesc	Same names and values as MQMD; see Table 163 on page 1754	-

Notes:

1. The symbol ¬ represents a single blank character.
2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.
3. In the C programming language, the macro variable MQXQH_DEFAULT contains the values listed above. Use it in the following way to provide initial values for the fields in the structure:

```
MQXQH MyXQH = {MQXQH_DEFAULT};
```

C declaration:

```
typedef struct tagMQXQH MQXQH;
struct tagMQXQH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQCHAR48  RemoteQName;     /* Name of destination queue */
  MQCHAR48  RemoteQMgrName;  /* Name of destination queue manager */
  MQMD1     MsgDesc;         /* Original message descriptor */
};
```

COBOL declaration:

```
**   MQXQH structure
  10 MQXQH.
**    Structure identifier
   15 MQXQH-STRUCID                  PIC X(4).
**    Structure version number
   15 MQXQH-VERSION                  PIC S9(9) BINARY.
**    Name of destination queue
   15 MQXQH-REMOTEQNAME              PIC X(48).
**    Name of destination queue manager
   15 MQXQH-REMOTEQMGRNAME           PIC X(48).
**    Original message descriptor
   15 MQXQH-MSGDESC.
**     Structure identifier
    20 MQXQH-MSGDESC-STRUCID          PIC X(4).
**     Structure version number
    20 MQXQH-MSGDESC-VERSION          PIC S9(9) BINARY.
**     Report options
    20 MQXQH-MSGDESC-REPORT           PIC S9(9) BINARY.
**     Message type
    20 MQXQH-MSGDESC-MSGTYPE          PIC S9(9) BINARY.
**     Expiry time
    20 MQXQH-MSGDESC-EXPIRY           PIC S9(9) BINARY.
**     Feedback or reason code
    20 MQXQH-MSGDESC-FEEDBACK         PIC S9(9) BINARY.
**     Numeric encoding of message data
    20 MQXQH-MSGDESC-ENCODING         PIC S9(9) BINARY.
**     Character set identifier of message data
    20 MQXQH-MSGDESC-CODEDCHARSETID   PIC S9(9) BINARY.
**     Format name of message data
    20 MQXQH-MSGDESC-FORMAT           PIC X(8).
**     Message priority
    20 MQXQH-MSGDESC-PRIORITY         PIC S9(9) BINARY.
**     Message persistence
    20 MQXQH-MSGDESC-PERSISTENCE      PIC S9(9) BINARY.
**     Message identifier
    20 MQXQH-MSGDESC-MSGID            PIC X(24).
**     Correlation identifier
    20 MQXQH-MSGDESC-CORRELID         PIC X(24).
**     Backout counter
    20 MQXQH-MSGDESC-BACKOUTCOUNT     PIC S9(9) BINARY.
**     Name of reply-to queue
    20 MQXQH-MSGDESC-REPLYTOQ         PIC X(48).
**     Name of reply queue manager
    20 MQXQH-MSGDESC-REPLYTOQMGR      PIC X(48).
**     User identifier
    20 MQXQH-MSGDESC-USERIDENTIFIER   PIC X(12).
**     Accounting token
    20 MQXQH-MSGDESC-ACCOUNTINGTOKEN  PIC X(32).
**     Application data relating to identity
    20 MQXQH-MSGDESC-APPLIDENTITYDATA PIC X(32).
**     Type of application that put the message
    20 MQXQH-MSGDESC-PUTAPPLTYPE      PIC S9(9) BINARY.
**     Name of application that put the message
    20 MQXQH-MSGDESC-PUTAPPLNAME      PIC X(28).
**     Date when message was put
```

```
    20 MQXQH-MSGDESC-PUTDATE          PIC X(8).
**     Time when message was put
    20 MQXQH-MSGDESC-PUTTIME          PIC X(8).
**     Application data relating to origin
    20 MQXQH-MSGDESC-APPLORIGINDATA   PIC X(4).
```

PL/I declaration:

```
dcl
 1 MQXQH based,
  3 StrucId           char(4),       /* Structure identifier */
  3 Version           fixed bin(31), /* Structure version number */
  3 RemoteQName       char(48),      /* Name of destination queue */
  3 RemoteQMgrName    char(48),      /* Name of destination queue
                                        manager */
  3 MsgDesc,                         /* Original message descriptor */
   5 StrucId          char(4),       /* Structure identifier */
   5 Version          fixed bin(31), /* Structure version number */
   5 Report           fixed bin(31), /* Report options */
   5 MsgType          fixed bin(31), /* Message type */
   5 Expiry           fixed bin(31), /* Expiry time */
   5 Feedback         fixed bin(31), /* Feedback or reason code */
   5 Encoding         fixed bin(31), /* Numeric encoding of message
                                        data */
   5 CodedCharSetId   fixed bin(31), /* Character set identifier of
                                        message data */
   5 Format           char(8),       /* Format name of message data */
   5 Priority         fixed bin(31), /* Message priority */
   5 Persistence      fixed bin(31), /* Message persistence */
   5 MsgId            char(24),      /* Message identifier */
   5 CorrelId         char(24),      /* Correlation identifier */
   5 BackoutCount     fixed bin(31), /* Backout counter */
   5 ReplyToQ         char(48),      /* Name of reply-to queue */
   5 ReplyToQMgr      char(48),      /* Name of reply queue manager */
   5 UserIdentifier   char(12),      /* User identifier */
   5 AccountingToken  char(32),      /* Accounting token */
   5 ApplIdentityData char(32),      /* Application data relating to
                                        identity */
   5 PutApplType      fixed bin(31), /* Type of application that put the
                                        message */
   5 PutApplName      char(28),      /* Name of application that put the
                                        message */
   5 PutDate          char(8),       /* Date when message was put */
   5 PutTime          char(8),       /* Time when message was put */
   5 ApplOriginData   char(4);       /* Application data relating to
                                        origin */
```

High Level Assembler declaration:

```
MQXQH                          DSECT
MQXQH_STRUCID                  DS   CL4   Structure identifier
MQXQH_VERSION                  DS   F     Structure version number
MQXQH_REMOTEQNAME              DS   CL48  Name of destination queue
MQXQH_REMOTEQMGRNAME           DS   CL48  Name of destination queue
*                                         manager
MQXQH_MSGDESC                  DS   0F    Force fullword alignment
MQXQH_MSGDESC_STRUCID          DS   CL4   Structure identifier
MQXQH_MSGDESC_VERSION          DS   F     Structure version number
MQXQH_MSGDESC_REPORT           DS   F     Report options
MQXQH_MSGDESC_MSGTYPE          DS   F     Message type
MQXQH_MSGDESC_EXPIRY           DS   F     Expiry time
MQXQH_MSGDESC_FEEDBACK         DS   F     Feedback or reason code
MQXQH_MSGDESC_ENCODING         DS   F     Numeric encoding of message
*                                         data
MQXQH_MSGDESC_CODEDCHARSETID   DS   F     Character set identifier of
*                                         message data
MQXQH_MSGDESC_FORMAT           DS   CL8   Format name of message data
MQXQH_MSGDESC_PRIORITY         DS   F     Message priority
```

```
MQXQH_MSGDESC_PERSISTENCE       DS   F     Message persistence
MQXQH_MSGDESC_MSGID             DS   XL24  Message identifier
MQXQH_MSGDESC_CORRELID          DS   XL24  Correlation identifier
MQXQH_MSGDESC_BACKOUTCOUNT      DS   F     Backout counter
MQXQH_MSGDESC_REPLYTOQ          DS   CL48  Name of reply-to queue
MQXQH_MSGDESC_REPLYTOQMGR       DS   CL48  Name of reply queue manager
MQXQH_MSGDESC_USERIDENTIFIER    DS   CL12  User identifier
MQXQH_MSGDESC_ACCOUNTINGTOKEN   DS   XL32  Accounting token
MQXQH_MSGDESC_APPLIDENTITYDATA  DS   CL32  Application data relating to
*                                          identity
MQXQH_MSGDESC_PUTAPPLTYPE       DS   F     Type of application that put
*                                          the message
MQXQH_MSGDESC_PUTAPPLNAME       DS   CL28  Name of application that put
*                                          the message
MQXQH_MSGDESC_PUTDATE           DS   CL8   Date when message was put
MQXQH_MSGDESC_PUTTIME           DS   CL8   Time when message was put
MQXQH_MSGDESC_APPLORIGINDATA    DS   CL4   Application data relating to
*                                          origin
MQXQH_MSGDESC_LENGTH            EQU  *-MQXQH_MSGDESC
                                ORG  MQXQH_MSGDESC
MQXQH_MSGDESC_AREA              DS   CL(MQXQH_MSGDESC_LENGTH)
*
MQXQH_LENGTH                    EQU  *-MQXQH
                                ORG  MQXQH
MQXQH_AREA                      DS   CL(MQXQH_LENGTH)
```

Visual Basic declaration:

```
Type MQXQH
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  RemoteQName    As String*48 'Name of destination queue'
  RemoteQMgrName As String*48 'Name of destination queue manager'
  MsgDesc        As MQMD1     'Original message descriptor'
End Type
```

Function calls

This section gives information on all of the MQI calls that are possible. Descriptions, syntax, parameter information, usage notes, and language invocations for each possible language are given for each of the different calls.

Call descriptions:

This section describes MQI calls.

- "MQGET - Get message" on page 1993
- "MQINQ - Inquire object attributes" on page 2005
- "MQINQMP - Inquire message property" on page 2020
- "MQMHBUF - Convert message handle into buffer" on page 2026
- "MQOPEN - Open object" on page 2030
- "MQPUT - Put message" on page 2047
- "MQPUT1 - Put one message" on page 2061
- "MQSET - Set object attributes" on page 2072
- "MQSETMP - Set message property" on page 2078
- "MQSTAT - Retrieve status information" on page 2082
- "MQMHBUF - Convert message handle into buffer" on page 2026
- "MQSUB - Register subscription" on page 2086
- "MQSUBRQ - Subscription request" on page 2093

Online help on the UNIX platforms, in the form of *man* pages, is available for these calls.

Note: The calls associated with data conversion, MQXCNVC and MQ_DATA_CONV_EXIT, are in "Data conversion" on page 2210.

Conventions used in the call descriptions:

For each call, this collection of topics gives a description of the parameters and usage of the call in a format that is independent of programming language. This is followed by typical invocations of the call, and typical declarations of its parameters, in each of the supported programming languages.

Important: When coding WebSphere MQ API calls you must ensure that all relevant parameters (as described in the following sections) are provided. Failure to do so can produce unpredictable results.

The description of each call contains the following sections:

Call name
The call name, followed by a brief description of the purpose of the call.

Parameters
For each parameter, the name is followed by its data type in parentheses ( ) and one of the following:

input You supply information in the parameter when you make the call.

output
The queue manager returns information in the parameter when the call completes or fails.

input/output
You supply information in the parameter when you make the call, and the queue manager changes the information when the call completes or fails.

For example:

Compcode (MQLONG) - output

In some cases, the data type is a structure. In all cases, there is more information about the data type or structure in "Elementary data types" on page 1530.

The last two parameters in each call are a completion code and a reason code. The completion code indicates whether the call completed successfully, partially, or not at all. Further information

about the partial success or the failure of the call is given in the reason code. For more information about each completion and reason code, see “Return codes” on page 2178.

Usage notes
Additional information about the call, describing how to use it and any restrictions on its use.

Assembler language invocation
Typical invocation of the call, and declaration of its parameters, in assembler language.

C invocation
Typical invocation of the call, and declaration of its parameters, in C.

COBOL invocation
Typical invocation of the call, and declaration of its parameters, in COBOL.

PL/I invocation
Typical invocation of the call, and declaration of its parameters, in PL/I.

All parameters are passed by reference.

Visual Basic invocation
Typical invocation of the call, and declaration of its parameters, in Visual Basic.

Other notation conventions are:

Constants
Names of constants are shown in uppercase; for example, MQOO_OUTPUT. A set of constants having the same prefix is shown as follows: MQIA_*. See “Constants” on page 1393 for the value of a constant.

Arrays
In some calls, parameters are arrays of character strings that do not have fixed sizes. In the descriptions of these parameters, a lowercase n represents a numeric constant. When you code the declaration for that parameter, replace the n with the numeric value that you require.

Using the calls in the C language:

Parameters that are *input only* and of type MQHCONN, MQHOBJ, MQHMSG, or MQLONG are passed by value. For all other parameters, the *address* of the parameter is passed by value.

You do not need to specify all parameters that are passed by address every time that you invoke a function. Where you do not need a particular parameter, specify a null pointer as the parameter on the function invocation, in place of the address of parameter data. Parameters for which this is possible are identified in the call descriptions.

No parameter is returned as the value of the call; in C terminology, this means that all calls return **void**.

Declaring the Buffer parameter:

The **MQGET**, **MQPUT**, and **MQPUT1** calls each have one parameter that has an undefined data type: the *Buffer* parameter. Use this parameter to send and receive the application's message data.

Parameters of this sort are shown in the C examples as arrays of MQBYTE. You can declare the parameters in this way, but it is usually more convenient to declare them as the particular structure that describes the layout of the data in the message. The function prototype declares the parameter as a pointer-to-void, so that you can specify the address of any sort of data as the parameter on the call invocation.

Pointer-to-void is a pointer to data of undefined format. It is defined as:

```
typedef void *PMQVOID;
```

MQBACK - Back out changes:

The MQBACK call indicates to the queue manager that all the message gets and puts that have occurred since the last sync point are to be backed out.

Messages put as part of a unit of work are deleted; messages retrieved as part of a unit of work are reinstated on the queue.

- On z/OS, this call is used only by batch programs (including IMS batch DL/I programs).
- On IBM i, this call is not supported for applications running in compatibility mode.

Syntax

MQBACK (*Hconn*, *Compcode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

Compcode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OUTCOME_MIXED
(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes

Usage notes

1. You can use this call only when the queue manager itself coordinates the unit of work. This can be:
   - A local unit of work, where the changes affect only MQ resources.
   - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting MQ resources.

   For further details about local and global units of work, see "MQBEGIN - Begin unit of work" on page 1931.
2. In environments where the queue manager does not coordinate the unit of work, use the appropriate back-out call instead of MQBACK. The environment might also support an implicit back out caused by the application terminating abnormally.
   - On z/OS, use the following calls:
     – Batch programs (including IMS batch DL/I programs) can use the MQBACK call if the unit of work affects only MQ resources. However, if the unit of work affects both MQ resources and resources belonging to other resource managers (for example, DB2), use the SRRBACK call provided by the z/OS Recoverable Resource Service (RRS). The SRRBACK call backs out changes to resources belonging to the resource managers that have been enabled for RRS coordination.
     – CICS applications must use the `EXEC CICS SYNCPOINT ROLLBACK` command to back out the unit of work. Do not use the MQBACK call for CICS applications.
     – IMS applications (other than batch DL/I programs) must use IMS calls such as `ROLB` to back out the unit of work. Do not use the MQBACK call for IMS applications (other than batch DL/I programs).
   - On IBM i, use this call for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the `STRCMTCTL` command with the `CMTSCOPE(*JOB)` parameter must not have been issued for the job.
3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC - Disconnect queue manager" on page 1984 for further details.

4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
   - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
   - Whether the message is part of a unit of work.
   - For the MQPUT call: whether the message is persistent or nonpersistent.

   The queue manager keeps *three* sets of group and segment information, one set for each of the following:
   - The last successful MQPUT call (this can be part of a unit of work).
   - The last successful MQGET call that removed a message from the queue (this can be part of a unit of work).
   - The last successful MQGET call that browsed a message on the queue (this *cannot* be part of a unit of work).
5. The information associated with the MQGET call is restored to the value that it had before the first successful MQGET call for that queue handle in the current unit of work.

   Queues that were updated by the application after the unit of work started, but outside the scope of the unit of work, do not have their group and segment information restored if the unit of work is backed out.

   Restoring the group and segment information to its previous value when a unit of work is backed out allows the application to spread a large message group or large logical message consisting of many segments across several units of work, and to restart at the correct point in the message group or logical message if one of the units of work fails.

   Using several units of work might be advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to be able to restart putting or getting messages at the correct point if a system failure occurs.

   For details of how to restart at the correct point after a system failure, see the MQPMO_LOGICAL_ORDER option described in "MQPMO - Put-message options" on page 1791, and the MQGMO_LOGICAL_ORDER option described in "MQGMO - Get-message options" on page 1655.

   The remaining usage notes apply only when the queue manager coordinates the units of work.
6. A unit of work has the same scope as a connection handle. All MQ calls that affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the *Hconn* parameter described in "MQCONN - Connect queue manager" on page 1961 for information about the scope of connection handles.
7. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
8. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but that never issues a commit or backout call, can fill queues with messages that are not available to other applications. To guard against this possibility, the administrator must set the *MaxUncommittedMsgs* queue-manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.

C invocation

```
MQBACK (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBACK' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBACK (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQBACK,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS  F  Connection handle
COMPCODE  DS  F  Completion code
REASON    DS  F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQBACK Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQBEGIN - Begin unit of work:

The MQBEGIN call begins a unit of work that is coordinated by the queue manager, and that can involve external resource managers.

Syntax

MQBEGIN (*Hconn*, *BeginOptions*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

Hconn must be a nonshared connection handle. If a shared connection handle is specified, the call fails with reason code MQRC_HCONN_ERROR. See the description of the MQCNO_HANDLE_SHARE_* options in "MQCNO - Connect options" on page 1607 for more information about shared and nonshared handles.

BeginOptions

Type: MQBO - input/output

These are options that control the action of MQBEGIN, as described in "MQBEGIN - Begin unit of work" on page 1931.

If no options are required, programs written in C or S/390 assembler can specify a null parameter address, instead of specifying the address of an MQBO structure.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_NO_EXTERNAL_PARTICIPANTS
(2121, X'849') No participating resource managers registered.

MQRC_PARTICIPANT_NOT_AVAILABLE
(2122, X'84A') Participating resource manager not available.

If *CompCode* is MQCC_FAILED:

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_BO_ERROR
(2134, X'856') Begin-options structure not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_IN_PROGRESS
(2128, X'850') Unit of work already started.

For more information about these reason codes, see Reason codes.

Usage notes

1. Use the MQBEGIN call to start a unit of work that is coordinated by the queue manager and that might involve changes to resources owned by other resource managers. The queue manager supports three types of unit-of-work:
   - **Queue-manager-coordinated local unit of work**: A unit of work in which the queue manager is the only resource manager participating, and so the queue manager acts as the unit-of-work coordinator.
     - To start this type of unit of work, specify the MQPMO_SYNCPOINT or MQGMO_SYNCPOINT option on the first MQPUT, MQPUT1, or MQGET call in the unit of work.
     - To commit or back out this type of unit of work, use the MQCMIT or MQBACK call.
   - **Queue-manager-coordinated global unit of work**: A unit of work in which the queue manager acts as the unit-of-work coordinator, both for MQ resources *and* for resources belonging to other resource managers. Those resource managers cooperate with the queue manager to ensure that all changes to resources in the unit of work are committed or backed out together.
     - To start this type of unit of work, use the MQBEGIN call.
     - To commit or back out this type of unit of work, use the MQCMIT and MQBACK calls.
   - **Externally-coordinated global unit of work**: A unit of work in which the queue manager is a participant, but the queue manager does not act as the unit-of-work coordinator. Instead, there is an external unit-of-work coordinator with which the queue manager cooperates.
     - To start this type of unit of work, use the relevant call provided by the external unit-of-work coordinator.

       If the MQBEGIN call is used to try to start the unit of work, the call fails with reason code MQRC_ENVIRONMENT_ERROR.
     - To commit or back out this type of unit of work, use the commit and back-out calls provided by the external unit-of-work coordinator.

       If you use the MQCMIT or MQBACK call to commit or back out the unit of work, the call fails with reason code MQRC_ENVIRONMENT_ERROR.
2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See the usage notes in "MQDISC - Disconnect queue manager" on page 1984 for further details.
3. An application can participate in only one unit of work at a time. The MQBEGIN call fails with reason code MQRC_UOW_IN_PROGRESS if there is already a unit of work in existence for the application, regardless of which type of unit of work it is.
4. The MQBEGIN call is not valid in an MQ MQI client environment. An attempt to use the call fails with reason code MQRC_ENVIRONMENT_ERROR.
5. When the queue manager is acting as the unit-of-work coordinator for global units of work, the resource managers that can participate in the unit of work are defined in the queue manager configuration file.
6. On IBM i, the three types of unit of work are supported as follows:
   - **Queue-manager-coordinated local unit of work** can be used only when a commitment definition does not exist at the job level, that is, the `STRCMTCTL` command with the `CMTSCOPE(*JOB)` parameter must not have been issued for the job.
   - **Queue-manager-coordinated global unit of work** is not supported.

- **Externally-coordinated global unit of work** can be used only when a commitment definition exists at job level, that is, the STRCMTCTL command with the CMTSCOPE(*JOB) parameter must have been issued for the job. If this has been done, the IBM i COMMIT and ROLLBACK operations apply to MQ resources as well as to resources belonging to other participating resource managers.

C invocation

```
MQBEGIN (Hconn, &BeginOptions, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQBO     BeginOptions;  /* Options that control the action of MQBEGIN */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQBEGIN' USING HCONN, BEGINOPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Options that control the action of MQBEGIN
 01  BEGINOPTIONS.
     COPY CMQBOV.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBEGIN (Hconn, BeginOptions, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl BeginOptions  like MQBO;      /* Options that control the action of
                                     MQBEGIN */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

Visual Basic invocation

```
MQBEGIN Hconn, BeginOptions, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long 'Connection handle'
Dim BeginOptions As MQBO 'Options that control the action of MQBEGIN'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

MQBUFMH - Convert buffer into message handle:

The MQBUFMH function call converts a buffer into a message handle and is the inverse of the MQMHBUF call.

This call takes a message descriptor and MQRFH2 properties in the buffer and makes them available through a message handle. The MQRFH2 properties in the message data are, optionally, removed. The *Encoding*, *CodedCharSetId*, and *Format* fields of the message descriptor are updated, if necessary, to correctly describe the contents of the buffer after the properties have been removed.

Syntax

MQBUFMH (*Hconn*, *Hmsg*, *BufMsgHOpts*, *MsgDesc*, *Buffer*, *BufferLength*, *DataLength*, *Compcode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread converting a buffer into a message handle. If a valid connection is not established, the call fails with MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMQSG - input

This is the message handle for which a buffer is required. The value was returned by a previous MQCRTMH call.

BufMsgHOpts
Type: MQBMHO - input

The MQBMHO structure allows applications to specify options that control how message handles are produced from buffers.

See "MQBMHO - Buffer to message handle options" on page 1562 for details.

MsgDesc
Type: MQMD - input/output

The *MsgDesc* structure contains the message descriptor properties and describes the contents of the buffer area.

On output from the call, the properties are optionally removed from the buffer area and, in this case, the message descriptor is updated to correctly describe the buffer area.

Data in this structure must be in the character set and encoding of the application.

BufferLength
Type: MQLONG - input

BufferLength is the length of the Buffer area, in bytes.

A *BufferLength* of zero bytes is valid, and indicates that the buffer area contains no data.

Buffer
Type: MQBYTExBufferLength - input/output

These are options that control the action of MQBEGIN, as described in "MQBEGIN - Begin unit of work" on page 1931.

Buffer defines the area containing the message buffer. For most data, you should align the buffer on a 4-byte boundary.

If *Buffer* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values appropriate to the data; this enables the data to be converted, if necessary.

If properties are found in the message buffer they are optionally removed; they later become available from the message handle on return from the call.

In the C programming language, the parameter is declared as a pointer-to-void, which means the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DataLength

Type: MQLONG - output

The length, in bytes, of the buffer which might have the properties removed.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BMHO_ERROR
(2489, X'09B9') Buffer to message handle options structure not valid.

MQRC_BUFFER_ERROR
(2004, X'07D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MD_ERROR
(2026, X'07EA') Message descriptor not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_RFH_ERROR
(2334, X'091E') MQRFH2 structure not valid.

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

MQBUFMH calls cannot be intercepted by API exits - a buffer is converted into a message handle in the application space; the call does not reach the queue manager.

C invocation

```
MQBUFMH (Hconn, Hmsg, &BufMsgHOpts, &MsgDesc, BufferLength, Buffer,
         &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;        /* Connection handle */
MQHMSG  Hmsg;         /* Message handle */
MQBMHO  BufMsgHOpts;  /* Options that control the action of MQBUFMH */
MQMD    MsgDesc;      /* Message descriptor */
MQLONG  BufferLength; /* Length in bytes of the Buffer area */
MQBYTE  Buffer[n];    /* Area to contain the message buffer */
MQLONG  DataLength;   /* Length of the output buffer */
MQLONG  CompCode;     /* Completion code */
MQLONG  Reason;       /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQBUFMH' USING HCONN, HMSG, BUFMSGHOPTS, MSGDESC, BUFFERLENGTH,
                     BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Message handle
 01  HMSG          PIC S9(18) BINARY.
**   Options that control the action of MQBUFMH
 01  BUFMSGHOPTS.
     COPY CMQBMHOV.
**   Message descriptor
 01  MSGDESC.
     COPY CMQMD.
**   Length in bytes of the Buffer area
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Area to contain the message buffer
 01  BUFFER        PIC X(n).
**   Length of the output buffer
```

```
 01  DATALENGTH    PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQBUFMH (Hconn, Hmsg, BufMsgHOpts, MsgDesc, BufferLength, Buffer,
              DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn        fixed bin(31); /* Connection handle */
dcl Hmsg         fixed bin(63); /* Message handle */
dcl BufMsgHOpts  like MQBMHO;   /* Options that control the action of
                                   MQBUFMH */
dcl MsgDesc      like MQMD;     /* Message descriptor */
dcl BufferLength fixed bin(31); /* Length in bytes of the Buffer area */
dcl Buffer       char(n);       /* Area to contain the message buffer */
dcl DataLength   fixed bin(31); /* Length of the output buffer */
dcl CompCode     fixed bin(31); /* Completion code */
dcl Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQBUFMH,(HCONN,HMSG,BUFMSGHOPTS,MSGDESC,BUFFERLENGTH,BUFFER,
                DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS       F      Connection handle
HMSG         DS       D      Message handle
BUFMSGHOPTS  CMQBMHOA ,      Options that control the action of MQBUFMH
MSGDESC      CMQMDA   ,      Message descriptor
BUFFERLENGTH DS       F      Length in bytes of the BUFFER area
BUFFER       DS       CL(n)  Area to contain the properties
DATALENGTH   DS       F      Length of the output buffer
COMPCODE     DS       F      Completion code
REASON       DS       F      Reason code qualifying COMPCODE
```

MQCB - Manage callback:

The MQCB call registers a callback for the specified object handle and controls activation and changes to the callback.

A callback is a piece of code (specified as either the name of a function that can be dynamically linked or as function pointer) that is called by IBM WebSphere MQ when certain events occur.

To use MQCB and MQCTL on a V7 client you must be connected to a V7 server and the **SHARECNV** parameter of the channel must have a non-zero value.

The types of callback that can be defined are:

Message consumer
A message consumer callback function is called when a message, meeting the selection criteria specified, is available on an object handle.

Only one callback function can be registered against each object handle. If a single queue is to be read with multiple selection criteria then the queue must be opened multiple times and a consumer function registered on each handle.

Event handler
The event handler is called for conditions that affect the whole callback environment.

The function is called when an event condition occurs, for example, a queue manager or connection stopping or quiescing.

The function is not called for conditions that are specific to a single message consumer, for example MQRC_GET_INHIBITED; it is called however if a callback function does not end normally.

Syntax

MQCB (*Hconn*, *Operation*, *CallbackDesc*, *Hobj*, *MsgDesc* , *GetMsgOpts*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, you can specify the following special value for *MQHC_DEF_HCONN* to use the connection handle associated with this execution unit.

Operation

Type: MQLONG - input

The operation being processed on the callback defined for the specified object handle. You must specify one of the following options; if more than one option is required, the values can be:

- Added (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

MQOP_REGISTER

Define the callback function for the specified object handle. This operation defines the function to be called and the selection criteria to be used.

If a callback function is already defined for the object handle the definition is replaced. If an error is detected while replacing the callback, the function is deregistered.

If a callback is registered in the same callback function in which it was previously deregistered, this is treated as a replace operation; any initial or final calls are not invoked.

You can use MQOP_REGISTER with MQOP_SUSPEND or MQOP_RESUME.

MQOP_DEREGISTER

Stop the consuming of messages for the object handle and removes the handle from those eligible for a callback.

A callback is automatically deregistered if the associated handle is closed.

If MQOP_DEREGISTER is called from within a consumer, and the callback has a stop call defined, it is invoked upon return from the consumer.

If this operation is issued against an *Hobj* with no registered consumer, the call returns with MQRC_CALLBACK_NOT_REGISTERED.

MQOP_SUSPEND

Suspends the consuming of messages for the object handle.

If this operation is applied to an event handler, the event handler does not get events while suspended, and any events missed while in the suspended state are not provided to the operation when it is resumed.

While suspended, the consumer function continues to get the control type callbacks.

MQOP_RESUME
Resume the consuming of messages for the object handle.

If this operation is applied to an event handler, the event handler does not get events while suspended, and any events missed while in the suspended state are not provided to the operation when it is resumed.

CallbackDesc
Type: MQCBD - input

This is a structure that identifies the callback function that is being registered by the application and the options used when registering it.

See MQCBD for details of the structure.

Callback descriptor is required only for the MQOP_REGISTER option; if the descriptor is not required, the parameter address passed can be null.

Hobj
Type: MQHOBJ - input

This handle represents the access that has been established to the object from which a message is to be consumed. This is a handle that has been returned from a previous MQOPEN or MQSUB call (in the *Hobj* parameter).

Hobj is not required when defining an event handler routine (MQCBT_EVENT_HANDLER) and should be specified as MQHO_NONE.

If *Hobj* has been returned from an MQOPEN call, the queue must have been opened with one or more of the following options:

- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_AS_Q_DEF
- MQOO_BROWSE

MsgDesc
Type: MQMD - input

This structure describes the attributes of the message required, and the attributes of the message retrieved.

The *MsgDesc* parameter defines the attributes of the messages required by the consumer, and the version of the MQMD to be passed to the message consumer.

The *MsgId*, *CorrelId*, *GroupId* , *MsgSeqNumber* , and *Offset* in the MQMD are used for message selection, depending on the options specified in the *GetMsgOpts* parameter.

The *Encoding* and *CodedCharSetId* are used for message conversion if you specify the MQGMO_CONVERT option.

See MQMD for details.

MsgDesc is used for MQOP_REGISTER and if you require values other than the default for any fields. *MsgDesc* is not used for an event handler.

If the descriptor is not required the parameter address passed can be null.

Note, that if multiple consumers are registered against the same queue with overlapping selectors, the chosen consumer for each message is undefined.

GetMsgOpts
Type: MQGMO - input

The *GetMsgOpts* parameter controls how the message consumer gets messages. All options of this parameter have meanings as described in "MQGMO - Get-message options" on page 1655, when used on an MQGET call, except:

MQGMO_SET_SIGNAL
This option is not permitted.

MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, MQGMO_MARK_*
The order of messages delivered to a browsing consumer is dictated by the combinations of these options. Significant combinations are:

MQGMO_BROWSE_FIRST
The first message on the queue is delivered repeatedly to the consumer. This is useful when the consumer destructively consumes the message in the callback. Use this option with care.

MQGMO_BROWSE_NEXT
The consumer is given each message on the queue, from the current cursor position until the end of the queue is reached.

MQGMO_BROWSE_FIRST + MQGMO_BROWSE_NEXT
The cursor is reset to the start of the queue. The consumer is then given each message until the cursor reaches the end of the queue.

MQGMO_BROWSE_FIRST + MQGMO_MARK_*
Starting at the beginning of the queue, the consumer is given the first nonmarked message on the queue, which is then marked for this consumer. This combination ensures that the consumer can receive new messages added behind the current cursor point.

MQGMO_BROWSE_NEXT + MQGMO_MARK_*
Starting at the cursor position, the consumer is given the next nonmarked message on the queue, which is then marked for this consumer. Use this combination with care because messages can be added to the queue behind the current cursor position.

MQGMO_BROWSE_FIRST + MQGMO_BROWSE_NEXT + MQGMO_MARK_*
This combination is not permitted. If used the call returns MQRC_OPTIONS_ERROR.

MQGMO_NO_WAIT, MQGMO_WAIT, and WaitInterval
These options control how the consumer is invoked.

MQGMO_NO_WAIT
The consumer is never called with MQRC_NO_MSG_AVAILABLE. The consumer is only called for messages and events.

MQGMO_WAIT with a zero WaitInterval
The MQRC_NO_MSG_AVAILABLE code is passed to the consumer when there are no messages available and either the consumer has been started or the consumer has been delivered at least one message since the last "no messages" reason code.

This prevents the consumer from polling in a busy loop when a zero wait interval is specified.

MQGMO_WAIT and a positive WaitInterval
The consumer is called after the specified wait interval with reason code MQRC_NO_MSG_AVAILABLE. This call is made regardless of whether any messages have been delivered to the consumer. This allows the user to perform heartbeat or batch type processing.

MQGMO_WAIT and WaitInterval of MQWI_UNLIMITED
This specifies an infinite wait before returning MQRC_NO_MSG_AVAILABLE. The consumer is never called with MQRC_NO_MSG_AVAILABLE.

GetMsgOpts is used only for MQOP_REGISTER and if you require values other than the default for any fields. *GetMsgOpts* is not used for an event handler.

If the *GetMsgOpts* are not required, the parameter address passed can be null. Using this parameter is the same as specifying MQGMO_DEFAULT together with MQGMO_FAIL_IF_QUIESCING.

If a message properties handle is provided in the MQGMO structure, a copy is provided in the MQGMO structure that is passed into the consumer callback. On return from the MQCB call, the application can delete the message properties handle.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason codes in the following list are the ones that the queue manager can return for the *Reason* parameter.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'855') Unable to load data conversion services modules.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CALLBACK_LINK_ERROR
(2487, X'9B7') Incorrect callback type field.

MQRC_CALLBACK_NOT_REGISTERED
(2448, X'990') Unable to unregister, suspend, or resume because there is no registered callback.

MQRC_CALLBACK_ROUTINE_ERROR
(2486, X'9B6') Either *CallbackFunction* or *CallbackName* must be specified but not both.

MQRC_CALLBACK_TYPE_ERROR
(2483, X'9B3') Incorrect callback type field.

MQRC_CBD_OPTIONS_ERROR
(2484, X'9B4') Incorrect MQCBD options field.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CORREL_ID_ERROR
(2207, X'89F') Correlation-identifier error.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_FUNCTION_NOT_SUPPORTED
(2298, X'8FA') The function requested is not available in the current environment.

MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_INVALID_MSG_UNDER_CURSOR
(2246, X'8C6') Message under cursor not valid for retrieval.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MATCH_OPTIONS_ERROR
(2247, X'8C7') Match options not valid.

MQRC_MAX_MSG_LENGTH_ERROR
(2485, X'9B4') Incorrect *MaxMsgLength* field.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MODULE_ENTRY_NOT_FOUND
(2497, X'9C1') The specified function entry point could not be found in the module.

MQRC_MODULE_INVALID
(2496, X'9C0') Module found, however it is of the wrong type; not 32 bit, 64 bit, or a valid dynamic link library.

MQRC_MODULE_NOT_FOUND
(2495, X'9BF') Module not found in the search path or not authorized to load.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Use of message token not valid.

MQRC_NO_MSG_AVAILABLE
(2033, X'7F1') No message available.

MQRC_NO_MSG_UNDER_CURSOR
(2034, X'7F2') Browse cursor not positioned on message.

MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'7F4') Queue not open for browse.

MQRC_NOT_OPEN_FOR_INPUT
(2037, X'7F5') Queue not open for input.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPERATION_ERROR
(2206, X'89E') Incorrect operation code on API Call.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_INDEX_TYPE_ERROR
(2394, X'95A') Queue has wrong index type.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SIGNAL_OUTSTANDING
(2069, X'815') Signal outstanding for this handle.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Sync point support not available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION
(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

For detailed information about these codes, see Reason codes.

Usage notes

1. MQCB is used to define the action to be invoked for each message, matching the specified criteria, available on the queue. When the action is processed, either the message is removed from the queue and passed to the defined message consumer, or a message token is provided, which is used to retrieve the message.
2. MQCB can be used to define callback routines before starting consumption with MQCTL or it can be used from within a callback routine.
3. To use MQCB from outside of a callback routine, you must first suspend message consumption by using MQCTL and resume consumption afterward.
4. MQCB is not supported within the IMS adapter.

Message consumer callback sequence

You can configure a consumer to invoke callback at key points during the lifecycle of the consumer. For example:

- when the consumer is first registered,
- when the connection is started,
- when the connection is stopped and
- when the consumer is deregistered, either explicitly, or implicitly by an MQCLOSE.

Table 209. MQCTL verb definitions

Verb	Meaning
MQCTL(START)	MQCTL call using the MQOP_START Operation
MQCTL(STOP)	MQCTL call using the MQOP_STOP Operation
MQCTL(WAIT)	MQCTL call using the MQOP_START_WAIT Operation

This allows the consumer to maintain state associated with the consumer. When a callback is requested by an application, the rules for consumer invocation are as follows:

REGISTER
Is always the first type of invocation of the callback.

Is always called on the same thread, as the MQCB(REGISTER) call.

START
Is always called synchronously with the MQCTL(START) verb.

- All START callbacks are completed before the MQCTL(START) verb returns.

Is on the same thread as the message delivery if THREAD_AFFINITY is requested.

The call with start is not guaranteed if, for example, a previous callback issues MQCTL(STOP) during the MQCTL(START).

STOP No further messages or events are delivered after this call until the connection is restarted.

A STOP is guaranteed if the application was previously called for START, or a message, or an event.

DEREGISTER
Is always the last type of invocation of the callback.

Ensure that your application performs thread-based initialization and cleanup in the START and STOP callbacks. You can do non-thread based initialization and cleanup with REGISTER and DEREGISTER callbacks.

Do not make any assumptions about the life and availability of the thread other than what is stated. For example, do not rely on a thread staying alive beyond the last call to DEREGISTER. Similarly, when you have chosen not to use THREAD_AFFINITY, do not assume that the thread exists whenever the connection is started.

If your application has particular requirements for thread characteristics, it can always create a thread accordingly, then use MQCTL(WAIT). This has the effect of 'donating' the thread to IBM WebSphere MQ for asynchronous message delivery.

Message consumer connection usage

You can configure a consumer to invoke callback at key points during the lifecycle of the consumer. For example:

- when the consumer is first registered,
- when the connection is started,
- when the connection is stopped and
- when the consumer is deregistered, either explicitly, or implicitly by an MQCLOSE.

Table 210. MQCTL verb definitions

Verb	Meaning
MQCTL(START)	MQCTL call using the MQOP_START Operation
MQCTL(STOP)	MQCTL call using the MQOP_STOP Operation
MQCTL(WAIT)	MQCTL call using the MQOP_START_WAIT Operation

This allows the consumer to maintain state associated with the consumer. When a callback is requested by an application, the rules for consumer invocation are as follows:

REGISTER
Is always the first type of invocation of the callback.

Is always called on the same thread, as the MQCB(REGISTER) call.

START
Is always called synchronously with the MQCTL(START) verb.

- All START callbacks are completed before the MQCTL(START) verb returns.

Is on the same thread as the message delivery if THREAD_AFFINITY is requested.

The call with start is not guaranteed if, for example, a previous callback issues MQCTL(STOP) during the MQCTL(START).

STOP No further messages or events are delivered after this call until the connection is restarted.

A STOP is guaranteed if the application was previously called for START, or a message, or an event.

DEREGISTER
Is always the last type of invocation of the callback.

Ensure that your application performs thread-based initialization and cleanup in the START and STOP callbacks. You can do non-thread based initialization and cleanup with REGISTER and DEREGISTER callbacks.

Do not make any assumptions about the life and availability of the thread other than what is stated. For example, do not rely on a thread staying alive beyond the last call to DEREGISTER. Similarly, when you have chosen not to use THREAD_AFFINITY, do not assume that the thread exists whenever the connection is started.

If your application has particular requirements for thread characteristics, it can always create a thread accordingly, then use MQCTL(WAIT). This has the effect of 'donating' the thread to IBM WebSphere MQ for asynchronous message delivery.

C invocation

```
MQCB (Hconn, Operation, CallbackDesc, Hobj, MsgDesc,
GetMsgOpts, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQLONG   Operation;     /* Operation being processed */
MQCBD    CallbackDesc;  /* Callback descriptor */
MQHOBJ   HObj           /* Object handle */
MQMD     MsgDesc        /* Message descriptor attributes */
MQGMO    GetMsgOpts     /* Message options */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCB' USING HCONN, OPERATION, CBDESC, HOBJ, MSGDESC,
            GETMSGOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Operation
 01  OPERATION PIC S9(9) BINARY.
**   Callback Descriptior
 01  CBDESC.
     COPY CMQCBDV.
01  HOBJ     PIC S9(9) BINARY.
**   Message Descriptior
 01  MSGDESC.
     COPY CMQMDV.
**   Get Message Options
 01  GETMSGOPTS.
     COPY CMQGMOV.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCB(Hconn, Operation, CallbackDesc, Hobj, MsgDesc, GetMsgOpts,
          CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn        fixed bin(31); /* Connection handle */
dcl Operation    fixed bin(31); /* Operation */
dcl CallbackDesc like MQCBD;    /* Callback Descriptor */
dcl Hobj         fixed bin(31); /* Object Handle */
dcl MsgDesc      like MQMD;     /* Message Descriptor */
dcl GetMsgOpts   like MQGMO;    /* Get Message Options */
dcl CompCode     fixed bin(31); /* Completion code */
dcl Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

MQCB_FUNCTION - Callback function:

The MQCB_FUNCTION function call is the callback function for event handling and asynchronous message consumption.

The MQCB_FUNCTION call definition is provided solely to describe the parameters that are passed to the callback function. No entry point called MQCB_FUNCTION is provided by the queue manager.

The specification of the actual function to be called is an input to the MQCB call and is passed in through the MQCBD structure.

Syntax

MQCB_FUNCTION (*Hconn*, *MsgDesc*, *GetMsgOpts*, *Buffer*, *Context*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call. On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for Hconn:

MQHC_DEF_CONN

Default connection handle.

MsgDesc

Type: MQMD - input

This structure describes the attributes of the message retrieved.

See "MQMD - Message descriptor" on page 1705 for details.

The version of MQMD passed is the same version as passed on the MQCB call that defined the consumer function.

The address of the MQMD is passed as null characters if a version 4 MQGMO was used to request that a Message Handle be returned instead of an MQMD.

This is an input field to the message consumer function; it is not relevant to an event handler function.

GetMsgOpts

Type: MQGMO - input

Options used to control the actions of the message consumer. This parameter also contains additional information about the message returned.

See MQGMO for details.

The version of MQGMO passed is the latest version supported.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Buffer

Type: MQBYTExBufferLength - input

This is the area containing the message data.

If no message is available for this call, or if the message contains no message data, the address of the *Buffer* is passed as nulls.

This is an input field to the message consumer function; it is not relevant to an event handler function.

Context

Type: MQCBC - input/output

This structure provides context information to the callback functions. See "MQCBC - Callback context" on page 1567 for details.

Usage notes

1. Be aware that if your callback routines use services that could delay or block the thread, for example, MQGET with wait, could delay the dispatch of other callbacks.
2. A separate unit of work is not automatically established for each invocation of a callback routine, so routines can either issue a commit call, or defer committing, until a logical batch of work has been processed. When the batch of work is committed, it commits the messages for all callback functions that have been invoked since the last sync point.
3. Programs invoked by CICS LINK or CICS START retrieve parameters using CICS services through named objects known as channel containers. The container names are the same as the parameter names. For more information, see your CICS documentation.

4. Callback routines can issue an MQDISC call, but not for their own connection. For example, if a callback routine has created a connection, then it can also disconnect the connection.
5. A callback routine should not, in general, rely on being invoked from the same thread each time. If required, use the MQCTLO_THREAD_AFFINITY when the connection is started.
6. When a callback routine receives a nonzero reason code, it must take appropriate action.
7. MQCB_FUNCTION is not supported within the IMS adapter.

MQCLOSE - Close object:

The MQCLOSE call relinquishes access to an object, and is the inverse of the MQOPEN and MQSUB calls.

Syntax

MQCLOSE (*Hconn*, *Hobj*, *Options*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, you can omit the MQCONN call, and specify the following value for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj
Type: MQHOBJ - input/output

This handle represents the object that is being closed. The object can be of any type. The value of *Hobj* was returned by a previous MQOPEN call.

On successful completion of the call, the queue manager sets this parameter to a value that is not a valid handle for the environment. This value is:

MQHO_UNUSABLE_HOBJ
Unusable object handle.

On z/OS, *Hobj* is set to a value that is undefined.

Options
Type: MQLONG - input

This parameter controls how the object is closed.

Only permanent dynamic queues and subscriptions can be closed in more than one way, because they must be either retained or deleted; these are queues with the *DefinitionType* attribute that has the value MQQDT_PERMANENT_DYNAMIC (see the *DefinitionType* attribute described in "Attributes for queues" on page 2135). The close options are summarized in this topic.

Durable subscriptions can either be kept or removed; these are created using the MQSUB call with the MQSO_DURABLE option.

When closing the handle to a managed destination (that is the *Hobj* parameter returned on an MQSUB call which used the MQSO_MANAGED option) the queue manager cleans up any publications that have not been retrieved when the associated subscription has also been removed.

The subscription is removed using the MQCO_REMOVE_SUB option on the *Hsub* parameter returned on an MQSUB call. Note MQCO_REMOVE_SUB is the default behavior on MQCLOSE for a non-durable subscription.

When closing a handle to a non-managed destination you are responsible for cleaning up the queue where publications are sent. Close the subscription using MQCO_REMOVE_SUB first and then process messages off the queue until there are none left.

You must specify one option only from the following:

Dynamic queue options: These options control how permanent dynamic queues are closed.

MQCO_DELETE
The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, created by a previous MQOPEN call, and there are no messages on the queue and no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned *Hobj*. In this case, all the messages on the queue are purged.

In all other cases, including the case where the *Hobj* was returned on an MQSUB call, the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

On z/OS, if the queue is a dynamic queue that has been logically deleted, and this is the last handle for it, the queue is physically deleted. See "Usage notes" on page 1955 for further details.

MQCO_DELETE_PURGE
The queue is deleted, and any messages on it purged, if either of the following is true:

- It is a permanent dynamic queue, created by a previous MQOPEN call, and there are no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned *Hobj*.

In all other cases, including the case where the *Hobj* was returned on an MQSUB call, the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

The table shows which close options are valid, and whether the object is retained or deleted.

Type of object or queue	MQCO_NONE	MQCO_DELETE	MQCO_DELETE_PURGE
Object other than a queue	Retained	Not valid	Not valid
Predefined queue	Retained	Not valid	Not valid
Permanent dynamic queue	Retained	Deleted if empty and no pending updates	Messages deleted; queue deleted if no pending updates
Temporary dynamic queue (call issued by creator of queue)	Deleted	Deleted	Deleted
Temporary dynamic queue (call not issued by creator of queue)	Retained	Not valid	Not valid
Distribution list	Retained	Not valid	Not valid
Managed subscription destination	Retained	Not valid	Not valid
Distribution list (subscription has been removed)	Messages deleted; queue deleted	Not valid	Not valid

Subscription closure options: These options control whether durable subscriptions are removed when the handle is closed, and whether publications still waiting to be read by the application are cleaned up. These options are only valid for use with an object handle returned in the *Hsub* parameter of an MQSUB call.

MQCO_KEEP_SUB
The handle to the subscription is closed but the subscription made is kept. Publications continue to be sent to the destination specified in the subscription. This option is only valid if the subscription was made with the option MQSO_DURABLE.

MQCO_KEEP_SUB is the default if the subscription is durable

MQCO_REMOVE_SUB
The subscription is removed and the handle to the subscription is closed.

The *Hobj* parameter of the MQSUB call is not invalidated by closure of the *Hsub* parameter and might continue to be used for MQGET or MQCB to receive the remaining publications. When the *Hobj* parameter of the MQSUB call is also closed, if it was a managed destination any unretrieved publications are removed.

MQCO_REMOVE_SUB is the default if the subscription is non-durable.

These subscription closure options are summarized in the following tables.

To close a durable subscription handle but retain the subscription, use the following subscription closure options:

Task	Subscription closure option
Keep publications on an MQOPENed handle	MQCO_KEEP_SUB
Remove publications on an MQOPENed handle	Action not allowed
Keep publications on an MQSO_MANAGED handle	MQCO_KEEP_SUB
Remove publications on an MQSO_MANAGED handle	Action not allowed

To unsubscribe, either by closing a durable subscription handle and unsubscribing it or closing a non-durable subscription handle, use the following subscription closure options:

Task	Subscription closure option
Keep publications on an MQOPENed handle	MQCO_REMOVE_SUB
Remove publications on an MQOPENed handle	Action not allowed
Keep publications on an MQSO_MANAGED handle	MQCO_REMOVE_SUB

Read ahead options: The following options control what happens to non-persistent messages which have been sent to the client before an application requested them and have not yet been consumed by the application. These messages are stored in the client read ahead buffer waiting to be requested by the application and can either be discarded or consumed from the queue before the MQCLOSE is completed.

MQCO_IMMEDIATE
The object is closed immediately and any messages which have been sent to the client before an application requested them are discarded and are not available to be consumed by any application. This is the default value.

MQCO_QUIESCE
A request to close the object is made, but if any messages which have been sent to the client before an application requested them, still reside in the client read ahead buffer, the MQCLOSE call returns with a warning of MQRC_READ_AHEAD_MSGS and the object handle remains valid.

The application can then continue to use the object handle to retrieve messages until no more are available, and then close the object again. No more messages are sent to the client ahead of an application requesting them, read ahead is now turned off.

Applications are advised to use MQCO_QUIESCE rather than trying to reach a point where there are no more messages in the client read ahead buffer, because a message could arrive between the last MQGET call and the following MQCLOSE which would be discarded if MQCO_IMMEDIATE was used.

If an MQCLOSE with MQCO_QUIESCE is issued from within an asynchronous callback function, the same behavior of reading ahead messages applies. If the warning MQRC_READ_AHEAD_MSGS is returned, then the callback function is called at least one more time. When the last remaining message that was read ahead has been passed to the callback function the MQCBC ConsumerFlags field is set to MQCBCF_READA_BUFFER_EMPTY.

Default option: If you require none of the options described above, you can use the following option:

MQCO_NONE
No optional close processing required.

This *must* be specified for:

- Objects other than queues
- Predefined queues
- Temporary dynamic queues (but only in those cases where *Hobj* is *not* the handle returned by the MQOPEN call that created the queue).
- Distribution lists

In all the above cases, the object is retained and not deleted.

If this option is specified for a temporary dynamic queue:

- The queue is deleted, if it was created by the MQOPEN call that returned *Hobj*; any messages that are on the queue are purged.
- In all other cases the queue (and any messages on it) are retained.

If this option is specified for a permanent dynamic queue, the queue is retained and not deleted.

On z/OS, if the queue is a dynamic queue that has been logically deleted, and this is the last handle for it, the queue is physically deleted. See “Usage notes” on page 1955 for further details.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason codes listed are the ones that the queue manager can return for the *Reason* parameter.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC__DB2_NOT_AVAILABLE
(2342, X'926') Db2 subsystem not available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPTION_NOT_VALID_FOR_TYPE
(2045, X'7FD') On an MQOPEN or MQCLOSE call: option not valid for object type.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

MQRC_READ_AHEAD_MSGS
(nnnn, X'xxx') The client has read ahead messages that have not yet been consumed by the application.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. When an application issues the MQDISC call, or ends either normally or abnormally, any objects that were opened by the application and are still open are closed automatically with the MQCO_NONE option.
2. The following points apply if the object being closed is a *queue*:
   - If operations on the queue are performed as part of a unit of work, the queue can be closed before or after the sync point occurs without affecting the outcome of the sync point. If the queue is triggered, performing a rollback before closing the queue can cause a trigger message to be issued. For more information about trigger messages, see Properties of trigger messages.
   - If the queue was opened with the MQOO_BROWSE option, the browse cursor is destroyed. If the queue is then reopened with the MQOO_BROWSE option, a new browse cursor is created (see MQOO_BROWSE).
   - If a message is currently locked for this handle at the time of the MQCLOSE call, the lock is released (see MQGMO_LOCK).
   - On z/OS, if there is an MQGET request with the MQGMO_SET_SIGNAL option outstanding against the queue handle being closed, the request is canceled (see MQGMO_SET_SIGNAL). Signal requests for the same queue but lodged against different handles (*Hobj*) are not affected (unless a dynamic queue is being deleted, in which case they are also canceled).
3. The following points apply if the object being closed is a *dynamic queue* (either permanent or temporary):

- For a dynamic queue, you can specify the MQCO_DELETE and MQCO_DELETE_PURGE options regardless of the options specified on the corresponding MQOPEN call.
- When a dynamic queue is deleted, all MQGET calls with the MQGMO_WAIT option that are outstanding against the queue are canceled and reason code MQRC_Q_DELETED is returned. See MQGMO_WAIT.

  Although applications cannot access a deleted queue, the queue is not removed from the system, and associated resources are not freed, until all handles that reference the queue have been closed, and all units of work that affect the queue have been either committed or backed out.

  On z/OS, a queue that has been logically deleted but not yet removed from the system prevents the creation of a new queue with the same name as the deleted queue; the MQOPEN call fails with reason code MQRC_NAME_IN_USE in this case. Also, such a queue can still be displayed using MQSC commands, even though it cannot be accessed by applications.
- When a permanent dynamic queue is deleted, if the *Hobj* handle specified on the MQCLOSE call is *not* the one that was returned by the MQOPEN call that created the queue, a check is made that the user identifier that was used to validate the MQOPEN call is authorized to delete the queue. If the MQOO_ALTERNATE_USER_AUTHORITY option was specified on the MQOPEN call, the user identifier checked is the *AlternateUserId*.

  This check is not performed if:
  - The handle specified is the one returned by the MQOPEN call that created the queue.
  - The queue being deleted is a temporary dynamic queue.
- When a temporary dynamic queue is closed, if the *Hobj* handle specified on the MQCLOSE call is the one that was returned by the MQOPEN call that created the queue, the queue is deleted. This occurs regardless of the close options specified on the MQCLOSE call. If there are messages on the queue, they are discarded; no report messages are generated.

  If there are uncommitted units of work that affect the queue, the queue and its messages are still deleted, but the units of work do not fail. However, as described above, the resources associated with the units of work are not freed until each of the units of work has been either committed or backed out.

4. The following points apply if the object being closed is a *distribution list*:
   - The only valid close option for a distribution list is MQCO_NONE; the call fails with reason code MQRC_OPTIONS_ERROR or MQRC_OPTION_NOT_VALID_FOR_TYPE if any other options are specified.
   - When a distribution list is closed, individual completion codes and reason codes are not returned for the queues in the list; only the *CompCode* and *Reason* parameters of the call are available for diagnostic purposes.

     If a failure occurs closing one of the queues, the queue manager continues processing and attempts to close the remaining queues in the distribution list. The *CompCode* and *Reason* parameters of the call are set to return information describing the failure. It is possible for the completion code to be MQCC_FAILED, even though most of the queues were closed successfully. The queue that encountered the error is not identified.

     If there is a failure on more than one queue, it is not defined which failure is reported in the *CompCode* and *Reason* parameters.
5. On IBM i, if the application was connected implicitly when the first MQOPEN call was issued, an implicit MQDISC occurs when the last MQCLOSE is issued.

   Only applications running in compatibility mode can be connected implicitly; other applications must issue the MQCONN or MQCONNX call to connect to the queue manager explicitly.

C invocation

```
MQCLOSE (Hconn, &Hobj, Options, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQHOBJ   Hobj;      /* Object handle */
MQLONG   Options;   /* Options that control the action of MQCLOSE */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQCLOSE' USING HCONN, HOBJ, OPTIONS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Object handle
 01  HOBJ      PIC S9(9) BINARY.
**   Options that control the action of MQCLOSE
 01  OPTIONS   PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCLOSE (Hconn, Hobj, Options, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl Hobj      fixed bin(31);  /* Object handle */
dcl Options   fixed bin(31);  /* Options that control the action of
                                 MQCLOSE */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQCLOSE,(HCONN,HOBJ,OPTIONS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS  F  Connection handle
HOBJ      DS  F  Object handle
OPTIONS   DS  F  Options that control the action of MQCLOSE
COMPCODE  DS  F  Completion code
REASON    DS  F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQCLOSE Hconn, Hobj, Options, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim Hobj     As Long 'Object handle'
Dim Options  As Long 'Options that control the action of MQCLOSE'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQCMIT - Commit changes:

The MQCMIT call indicates to the queue manager that the application has reached a sync point, and that all the message gets and puts that have occurred since the last sync point are to be made permanent.

Messages put as part of a unit of work are made available to other applications; messages retrieved as part of a unit of work are deleted.

- On z/OS, the call is used only by batch programs (including IMS batch DL/I programs).
- On IBM i, this call is not supported for applications running in compatibility mode.

Syntax

MQCMIT (*Hconn*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason codes listed are the ones that the queue manager can return for the *Reason* parameter.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_OUTCOME_PENDING
(2124, X'84C') Result of commit operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CALL_INTERRUPTED
(2549, X'9F5') MQPUT or MQCMIT was interrupted and reconnection processing cannot reestablish a definite outcome.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OUTCOME_MIXED
(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RECONNECT_FAILED
(2548, X'9F4') After reconnecting, an error occurred reinstating the handles for a reconnectable connection.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. Use this call only when the queue manager itself coordinates the unit of work. This can be:
   - A local unit of work, where the changes affect only WebSphere MQ resources.
   - A global unit of work, where the changes can affect resources belonging to other resource managers, as well as affecting WebSphere MQ resources.

   For further details about local and global units of work, see "MQBEGIN - Begin unit of work" on page 1931.
2. In environments where the queue manager does not coordinate the unit of work, the appropriate commit call must be used instead of MQCMIT. The environment might also support an implicit commit caused by the application terminating normally.
   - On z/OS, use the following calls:
     – Batch programs (including IMS batch DL/I programs) can use the MQCMIT call if the unit of work affects only WebSphere MQ resources. However, if the unit of work affects both WebSphere MQ resources and resources belonging to other resource managers (for example, DB2), use the

SRRCMIT call provided by the z/OS Recoverable Resource Service (RRS). The SRRCMIT call commits changes to resources belonging to the resource managers that have been enabled for RRS coordination.

– CICS applications must use the `EXEC CICS SYNCPOINT` command to commit the unit of work explicitly. Alternatively, ending the transaction results in an implicit commit of the unit of work. The MQCMIT call cannot be used for CICS applications.

– IMS applications (other than batch DL/I programs) must use IMS calls such as `GU` and `CHKP` to commit the unit of work. The MQCMIT call cannot be used for IMS applications (other than batch DL/I programs).

- On IBM i, use this call for local units of work coordinated by the queue manager. This means that a commitment definition must not exist at job level, that is, the `STRCMTCTL` command with the `CMTSCOPE(*JOB)` parameter must not have been issued for the job.

3. If an application ends with uncommitted changes in a unit of work, the disposition of those changes depends on whether the application ends normally or abnormally. See MQDISC usage notes for further details.
4. When an application puts or gets messages in groups or segments of logical messages, the queue manager retains information relating to the message group and logical message for the last successful MQPUT and MQGET calls. This information is associated with the queue handle, and includes such things as:
   - The values of the *GroupId*, *MsgSeqNumber*, *Offset*, and *MsgFlags* fields in MQMD.
   - Whether the message is part of a unit of work.
   - For the MQPUT call: whether the message is persistent or nonpersistent.

   When a unit of work is committed, the queue manager retains the group and segment information, and the application can continue putting or getting messages in the current message group or logical message.

   Retaining the group and segment information when a unit of work is committed allows the application to spread a large message group or large logical message consisting of many segments across several units of work. Using several units of work is advantageous if the local queue manager has only limited queue storage. However, the application must maintain sufficient information to restart putting or getting messages at the correct point if a system failure occurs. For details of how to restart at the correct point after a system failure, see MQPMO_LOGICAL_ORDER and MQGMO_LOGICAL_ORDER.

   The remaining usage notes apply only when the queue manager coordinates the units of work:
5. A unit of work has the same scope as a connection handle; all WebSphere MQ calls that affect a particular unit of work must be performed using the same connection handle. Calls issued using a different connection handle (for example, calls issued by another application) affect a different unit of work. See the *Hconn* parameter described in MQCONN for information about the scope of connection handles.
6. Only messages that were put or retrieved as part of the current unit of work are affected by this call.
7. A long-running application that issues MQGET, MQPUT, or MQPUT1 calls within a unit of work, but that never issues a commit or back-out call, can fill queues with messages that are not available to other applications. To guard against this, the administrator must set the *MaxUncommittedMsgs* queue-manager attribute to a value that is low enough to prevent runaway applications filling the queues, but high enough to allow the expected messaging applications to work correctly.
8. On UNIX and Windows systems, if the *Reason* parameter is MQRC_CONNECTION_BROKEN (with a *CompCode* of MQCC_FAILED), or MQRC_UNEXPECTED_ERROR it is possible that the unit of work was successfully committed.

C invocation

```
MQCMIT (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCMIT' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCMIT (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQCMIT,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS  F  Connection handle
COMPCODE  DS  F  Completion code
REASON    DS  F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQCMIT Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQCONN - Connect queue manager:

The MQCONN call connects an application program to a queue manager.

It provides a queue manager connection handle, which the application uses on subsequent message queuing calls.

- On z/OS, CICS applications do not have to issue this call. These applications are connected automatically to the queue manager to which the CICS system is connected. However, the MQCONN and MQDISC calls are still accepted from CICS applications.
- On IBM i, applications running in compatibility mode do not have to issue this call. These applications are connected automatically to the queue manager when they issue the first MQOPEN call. However, the MQCONN and MQDISC calls are still accepted from IBM i applications.

  Other applications (that is, applications not running in compatibility mode) must use the MQCONN or MQCONNX call to connect to the queue manager, and the MQDISC call to disconnect from the queue manager. This is the recommended style of programming.

A client connection cannot be made on a server only installation, and a local connection cannot be made on a client only installation.

Syntax

MQCONN (*QMgrName*, *Hconn*, *CompCode*, *Reason*)

Parameters

QMgrName

Type: MQCHAR48 - input

This is the name of the queue manager to which the application wants to connect. The name can contain the following characters:

- Uppercase alphabetic characters (A through Z)
- Lowercase alphabetic characters (a through z)
- Numeric digits (0 through 9)
- Period (.), forward slash (/), underscore (_), percent (%)

The name must not contain leading or embedded blanks, but can contain trailing blanks. A null character can be used to indicate the end of significant data in the name; the null and any characters following it are treated as blanks. The following restrictions apply in the environments indicated:

- On systems that use EBCDIC Katakana, lowercase characters cannot be used.
- On z/OS, names that begin or end with an underscore cannot be processed by the operations and control panels. For this reason, avoid such names.
- On IBM i, enclose names containing lowercase characters, forward slash, or percent in quotation marks when specified on commands. Do not specify these quotation marks in the *QMgrName* parameter.

If the name consists entirely of blanks, the name of the *default* queue manager is used.

The name specified for *QMgrName* must be the name of a *connectable* queue manager.

On z/OS, the queue managers to which it is possible to connect are determined by the environment:

- For CICS, you can use only the queue manager to which the CICS system is connected. The *QMgrName* parameter must still be specified, but its value is ignored; blanks are recommended.
- For IMS, only queue managers that are listed in the subsystem definition table (CSQQDEFV), *and* listed in the SSM table in IMS, are connectable (see usage note 6).
- For z/OS batch and TSO, only queue managers that reside on the same system as the application are connectable (see usage note 6).

Queue-sharing groups: On systems where several queue managers exist and are configured to form a queue-sharing group, the name of the queue-sharing group can be specified for *QMgrName* in place of the name of a queue manager. This allows the application to connect to *any* queue manager that is available in the queue-sharing group and that is on the same z/OS image as the application. The system can also be configured so that using a blank *QMgrName* connects to the queue-sharing group instead of to the default queue manager.

If *QMgrName* specifies the name of the queue-sharing group, but there is also a queue manager with that name on the system, connection is made to the latter in preference to the former. Only if that connection fails is connection to one of the queue managers in the queue-sharing group attempted.

If the connection is successful, you can use the handle returned by the MQCONN or MQCONNX call to access *all* the resources (both shared and nonshared) that belong to the queue manager to which connection has been made. Access to these resources is subject to the typical authorization controls.

If the application issues two MQCONN or MQCONNX calls to establish concurrent connections, and one or both calls specifies the name of the queue-sharing group, the second call returns completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED when it connects to the same queue manager as the first call.

Queue-sharing groups are supported only on z/OS. Connection to a queue-sharing group is supported only in the batch, RRS batch, and TSO environments.

WebSphere MQ MQI client applications: For WebSphere MQ MQI client applications, a connection is attempted for each client-connection channel definition with the specified queue-manager name, until one is successful. The queue manager, however, must have the same name as the specified name. If an all-blank name is specified, each client-connection channel with an all-blank queue-manager name is tried until one is successful; in this case there is no check against the actual name of the queue manager.

WebSphere MQ client applications are not supported in z/OS, but z/OS can act as an WebSphere MQ server, to which WebSphere MQ client applications can connect.

WebSphere MQ MQI client queue-manager groups: If the specified name starts with an asterisk (*), the queue manager to which connection is made might have a different name from that specified by the application. The specified name (without the asterisk) defines a *group* of queue managers that are eligible for connection. The implementation selects one from the group by trying each one in turn until one is found to which a connection can be made. The order in which connections are attempted is influenced by the client channel weight and connection affinity values of the candidate channels. If none of the queue managers in the group is available for connection, the call fails. Each queue manager is tried once only. If an asterisk alone is specified for the name, an implementation-defined default queue-manager group is used.

Queue-manager groups are supported only for applications running in an MQ-client environment; the call fails if a non-client application specifies a queue-manager name beginning with an asterisk. A group is defined by providing several client connection channel definitions with the same queue-manager name (the specified name without the asterisk), to communicate with each of the queue managers in the group. The default group is defined by providing one or more client connection channel definitions, each with a blank queue-manager name (specifying an all-blank name therefore has the same effect as specifying a single asterisk for the name for a client application).

After connecting to one queue manager of a group, an application can specify blanks in the typical way in the queue-manager name fields in the message and object descriptors to mean the name of the queue manager to which the application has connected (the *local queue manager*). If the application needs to know this name, use the MQINQ call to inquire the *QMgrName* queue-manager attribute.

Prefixing an asterisk to the connection name implies that the application does not depend on connecting to a particular queue manager in the group. Suitable applications are:

- Applications that put messages but do not get messages.
- Applications that put request messages and then get the reply messages from a *temporary dynamic* queue.

Unsuitable applications are ones that need to get messages from a particular queue at a particular queue manager; such applications must not prefix the name with an asterisk.

If you specify an asterisk, the maximum length of the remainder of the name is 47 characters.

Queue-manager groups are not supported on z/OS.

The length of this parameter is given by MQ_Q_MGR_NAME_LENGTH.

Hconn

Type: MQHCONN - output

This handle represents the connection to the queue manager. Specify it on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued, or when the unit of processing that defines the scope of the handle terminates.

WebSphere MQ now supplies the mqm library with client packages as well as server packages. This means that when an MQI call that is found in the mqm library is made, the connection type is

checked to see if it is a client or server connection, and then the correct underlying call is made. Therefore an exit which is passed an *Hconn* can now be linked against the mqm library, but used on a client installation.

Handle scope: The scope of the handle returned depends on the call used to connect to the queue manager (MQCONN or MQCONNX). If the call used is MQCONNX, the scope of the handle also depends on the MQCNO_HANDLE_SHARE_* option specified in the *Options* field of the MQCNO structure.

- If the call is MQCONN, or the MQCNO_HANDLE_SHARE_NONE option is specified, the handle returned is a *nonshared* handle.

  The scope of a nonshared handle is the smallest unit of parallel processing supported by the platform on which the application is running (see Table 211 for details); the handle is not valid outside the unit of parallel processing from which the call was issued.
- If you specify the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK option, the handle returned is a *shared* handle.

  The scope of a shared handle is the process that owns the thread from which the call was issued; the handle can be used from any thread that belongs to that process. Not all platforms support threads.
- If the MQCONN or MQCONNX call fails with completion code equal to MQCC_FAILED, then the Hconn value is undefined.

Table 211. Scope of nonshared handles on various platforms

Platform	Scope of nonshared handle
z/OS	• CICS: the CICS task • IMS: the task, up to the next sync point (excluding subtasks of the task) • z/OS batch and TSO: the task (excluding subtasks of the task)
IBM i	Job
UNIX systems	Thread
16 bit Windows applications	Process
32 bit Windows applications	Thread

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the value returned is:

MQHC_DEF_HCONN
Default connection handle.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_ALREADY_CONNECTED
(2002, X'7D2') Application already connected.

MQRC_CLUSTER_EXIT_LOAD_ERROR
(2267, X'8DB') Unable to load cluster workload exit.

MQRC_SSL_ALREADY_INITIALIZED
(2391, X'957') SSL already initialized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_CONN_LOAD_ERROR
(2129, X'851') Unable to load adapter connection module.

MQRC_ADAPTER_DEFS_ERROR
(2131, X'853') Adapter subsystem definition module not valid.

MQRC_ADAPTER_DEFS_LOAD_ERROR
(2132, X'854') Unable to load adapter subsystem definition module.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ADAPTER_STORAGE_SHORTAGE
(2127, X'84F') Insufficient storage for adapter.

MQRC_ANOTHER_Q_MGR_CONNECTED
(2103, X'837') Another queue manager already connected.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR
(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR
(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONN_ID_IN_USE
(2160, X'870') Connection identifier already in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_ERROR
(2273, X'8E1') Error processing MQCONN call.

MQRC_CONNECTION_NOT_AVAILABLE
(2568, X'A08') Occurs on an MQCONN or MQCONNX call when the queue manager is

unable to provide a connection of the requested connection type on the current installation. A client connection cannot be made on a server only installation. A local connection cannot be made on a client only installation.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CRYPTO_HARDWARE_ERROR
(2382, X'94E') Cryptographic hardware configuration error.

MQRC_DUPLICATE_RECOV_COORD
(2163, X'873') Recovery coordinator exists.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOST_NOT_AVAILABLE
(2538, X'9EA') An MQCONN call was issued from a client to connect to a queue manager but the attempt to allocate a conversation to the remote system failed.

MQRC_INSTALLATION_MISMATCH
(2583, X'A17') Mismatch between queue manager installation and selected library.

MQRC_KEY_REPOSITORY_ERROR
(2381, X'94D') Key repository not valid.

MQRC_MAX_CONNS_LIMIT_REACHED
(2025, X'7E9') Maximum number of connections reached.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OPEN_FAILED
(2137, X'859') Object not opened successfully.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SSL_INITIALIZATION_ERROR
(2393, X'959') SSL initialization error.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. The queue manager to which connection is made using the MQCONN call is called the *local queue manager*.
2. Queues that are owned by the local queue manager appear to the application as local queues. It is possible to put messages on and get messages from these queues.

   Shared queues that are owned by the queue-sharing group to which the local queue manager belongs appear to the application as local queues. It is possible to put messages on and get messages from these queues.

   Queues that are owned by remote queue managers appear as remote queues. It is possible to put messages on these queues, but not to get messages from these queues.
3. If the queue manager fails while an application is running, the application must issue the MQCONN call again to obtain a new connection handle to use on subsequent WebSphere MQ calls. The application can issue the MQCONN call periodically until the call succeeds.

   If an application is not sure whether it is connected to the queue manager, the application can safely issue an MQCONN call to obtain a connection handle. If the application is already connected, the handle returned is the same as that returned by the previous MQCONN call, but with completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED.
4. When the application has finished using WebSphere MQ calls, the application must use the MQDISC call to disconnect from the queue manager.
5. If the MQCONN call fails with completion code equal to MQCC_FAILED, then the Hconn value is undefined.
6. On z/OS:
   - Batch, TSO, and IMS applications must issue the MQCONN call to use the other WebSphere MQ calls. These applications can connect to more than one queue manager concurrently.

     If the queue manager fails, the application must issue the call again after the queue manager has restarted to obtain a new connection handle.

     Although IMS applications can issue the MQCONN call repeatedly, even when already connected, this is not recommended for online message processing programs (MPPs).
   - CICS applications do not have to issue the MQCONN call to use the other WebSphere MQ calls, but can do so if they want; both the MQCONN call and the MQDISC call are accepted. However, it is not possible to connect to more than one queue manager concurrently.

     If the queue manager fails, these applications are automatically reconnected when the queue manager restarts, and so do not need to issue the MQCONN call.
7. On z/OS, to define the available queue managers:
   - For batch applications, system programmers can use the CSQBDEF macro to create a module (CSQBDEFV) that defines the default queue-manager name, or queue-sharing group name.
   - For IMS applications, system programmers can use the CSQQDEFX macro to create a module (CSQQDEFV) that defines the names of the available queue managers and specifies the default queue manager.

     In addition, each queue manager must be defined to the IMS control region and to each dependent region accessing that queue manager. To do this, you must create a subsystem member in the IMS.PROCLIB library and identify the subsystem member to the applicable IMS regions. If an application attempts to connect to a queue manager that is not defined in the subsystem member for its IMS region, the application abends.
8. On IBM i, applications written for previous releases of the queue manager can run without recompiling. This is called *compatibility mode*. This mode of operation provides a compatible runtime environment for applications. It comprises the following:

- The service program AMQZSTUB residing in the library QMQM.

  AMQZSTUB provides the same public interface as previous releases, and has the same signature. Use this service program to access the MQI through bound procedure calls.
- The program QMQM residing in the library QMQM.

  QMQM provides a means of accessing the MQI through dynamic program calls.
- Programs MQCLOSE, MQCONN, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, and MQSET residing in the library QMQM.

  These programs also provide a means of accessing the MQI through dynamic program calls, but with a parameter list that corresponds to the standard descriptions of the WebSphere MQ calls.

These three interfaces do not include capabilities that were introduced in WebSphere MQ Version 5.1. For example, the MQBACK, MQCMIT, and MQCONNX calls are not supported. The support provided by these interfaces is for single-threaded applications only.

Support for the new WebSphere MQ calls in single-threaded applications, and for all WebSphere MQ calls in multi-threaded applications, is provided through the service programs LIBMQM and LIBMQM_R.

9. On IBM i, programs that end abnormally are not automatically disconnected from the queue manager. Write applications to allow for the possibility of the MQCONN or MQCONNX call returning completion code MQCC_WARNING and reason code MQRC_ALREADY_CONNECTED. Use the connection handle returned in this situation as normal.

C invocation

```
MQCONN (QMgrName, &Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;  /* Name of queue manager */
MQHCONN   Hconn;     /* Connection handle */
MQLONG    CompCode;  /* Completion code */
MQLONG    Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONN' USING QMGRNAME, HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Name of queue manager
 01  QMGRNAME  PIC X(48).
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCONN (QMgrName, Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl QMgrName  char(48);       /* Name of queue manager */
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQCONN,(QMGRNAME,HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
QMGRNAME  DS  CL48  Name of queue manager
HCONN     DS  F     Connection handle
COMPCODE  DS  F     Completion code
REASON    DS  F     Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQCONN QMgrName, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim QMgrName As String*48 'Name of queue manager'
Dim Hconn    As Long      'Connection handle'
Dim CompCode As Long      'Completion code'
Dim Reason   As Long      'Reason code qualifying CompCode'
```

MQCONNX - Connect queue manager (extended):

The MQCONNX call connects an application program to a queue manager. It provides a queue manager connection handle, which is used by the application on subsequent WebSphere MQ calls.

The MQCONNX call is like the MQCONN call, except that MQCONNX allows options to be specified to control the way that the call works.

- This call is supported on all WebSphere MQ systems, and WebSphere MQ clients connected to these systems.
- On IBM i, this call is not supported for applications running in compatibility mode.

A client connection cannot be made on a server only installation, and a local connection cannot be made on a client only installation.

Syntax

MQCONNX (*QMgrName*, *ConnectOpts*, *Hconn*, *CompCode*, *Reason*)

Parameters

QMgrName

Type: MQCHAR48 - input

See the *QMgrName* parameter described in "MQCONN - Connect queue manager" on page 1961 for details.

ConnectOpts

Type: MQCNO - input/output

See "MQCNO - Connect options" on page 1607 for details.

Hconn

Type: MQHCONN - output

This handle represents the connection to the queue manager. Specify it on all subsequent message queuing calls issued by the application. It ceases to be valid when the MQDISC call is issued, or when the unit of processing that defines the scope of the handle terminates.

WebSphere MQ now supplies the mqm library with client packages as well as server packages. This means that when an MQI call that is found in the mqm library is made, the connection type is checked to see if it is a client or server connection, and then the correct underlying call is made. Therefore an exit which is passed an *Hconn* can now be linked against the mqm library, but used on a client installation.

Handle scope: The scope of the handle returned depends on the call used to connect to the queue manager (MQCONN or MQCONNX). If the call used is MQCONNX, the scope of the handle also depends on the MQCNO_HANDLE_SHARE_* option specified in the *Options* field of the MQCNO structure.

- If the call is MQCONN, or the MQCNO_HANDLE_SHARE_NONE option is specified, the handle returned is a *nonshared* handle.

  The scope of a nonshared handle is the smallest unit of parallel processing supported by the platform on which the application is running (see Table 212 for details); the handle is not valid outside the unit of parallel processing from which the call was issued.
- If you specify the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK option, the handle returned is a *shared* handle.

  The scope of a shared handle is the process that owns the thread from which the call was issued; the handle can be used from any thread that belongs to that process. Not all platforms support threads.
- If the MQCONN or MQCONNX call fails with completion code equal to MQCC_FAILED, then the Hconn value is undefined.

Table 212. Scope of nonshared handles on various platforms

Platform	Scope of nonshared handle
z/OS	• CICS: the CICS task • IMS: the task, up to the next sync point (excluding subtasks of the task) • z/OS batch and TSO: the task (excluding subtasks of the task)
IBM i	Job
UNIX systems	Thread
16 bit Windows applications	Process
32 bit Windows applications	Thread

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the value returned is:

MQHC_DEF_HCONN
Default connection handle.

CompCode
Type: MQLONG - output

See the *CompCode* parameter described in "MQCONN - Connect queue manager" on page 1961 for details.

Reason
Type: MQLONG - output

The following codes can be returned by the MQCONN and MQCONNX calls. For a list of additional codes that can be returned by the MQCONNX call, see the following codes.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_ALREADY_CONNECTED
(2002, X'7D2') Application already connected.

MQRC_CLUSTER_EXIT_LOAD_ERROR
(2267, X'8DB') Unable to load cluster workload exit.

MQRC_SSL_ALREADY_INITIALIZED
(2391, X'957') SSL already initialized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_CONN_LOAD_ERROR
(2129, X'851') Unable to load adapter connection module.

MQRC_ADAPTER_DEFS_ERROR
(2131, X'853') Adapter subsystem definition module not valid.

MQRC_ADAPTER_DEFS_LOAD_ERROR
(2132, X'854') Unable to load adapter subsystem definition module.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ADAPTER_STORAGE_SHORTAGE
(2127, X'84F') Insufficient storage for adapter.

MQRC_ANOTHER_Q_MGR_CONNECTED
(2103, X'837') Another queue manager already connected.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR
(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR
(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONN_ID_IN_USE
(2160, X'870') Connection identifier already in use.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_ERROR
(2273, X'8E1') Error processing MQCONN call.

MQRC_CONNECTION_NOT_AVAILABLE
(2568, X'A08') Occurs on an MQCONN or MQCONNX call when the queue manager is unable to provide a connection of the requested connection type on the current installation. A client connection cannot be made on a server only installation. A local connection cannot be made on a client only installation.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CRYPTO_HARDWARE_ERROR
(2382, X'94E') Cryptographic hardware configuration error.

MQRC_DUPLICATE_RECOV_COORD
(2163, X'873') Recovery coordinator exists.

MQRC_ENVIRONMENT_ERROR
(2012, X'7DC') Call not valid in environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOST_NOT_AVAILABLE
(2538, X'9EA') An MQCONN call was issued from a client to connect to a queue manager but the attempt to allocate a conversation to the remote system failed.

MQRC_INSTALLATION_MISMATCH
(2583, X'A17') Mismatch between queue manager installation and selected library.

MQRC_KEY_REPOSITORY_ERROR
(2381, X'94D') Key repository not valid.

MQRC_MAX_CONNS_LIMIT_REACHED
(2025, X'7E9') Maximum number of connections reached.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OPEN_FAILED
(2137, X'859') Object not opened successfully.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SSL_INITIALIZATION_ERROR
(2393, X'959') SSL initialization error.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

The following additional reason codes can be returned by the MQCONNX call:

If *CompCode* is MQCC_FAILED:

MQRC_AIR_ERROR
(2385, X'951') Authentication information record not valid.

MQRC_AUTH_INFO_CONN_NAME_ERROR
(2387, X'953') Authentication information connection name not valid.

MQRC_AUTH_INFO_REC_COUNT_ERROR
(2383, X'94F') Authentication information record count not valid.

MQRC_AUTH_INFO_REC_ERROR
(2384, X'950') Authentication information record fields not valid.

MQRC_AUTH_INFO_TYPE_ERROR
(2386, X'952') Authentication information type not valid.

MQRC_CD_ERROR
(2277, X'8E5') Channel definition not valid.

MQRC_CLIENT_CONN_ERROR
(2278, X'8E6') Client connection fields not valid.

MQRC_CNO_ERROR
(2139, X'85B') Connect-options structure not valid.

MQRC_CONN_TAG_IN_USE
(2271, X'8DF') Connection tag in use.

MQRC_CONN_TAG_NOT_USABLE
(2350, X'92E') Connection tag not usable.

MQRC_LDAP_PASSWORD_ERROR
(2390, X'956') LDAP password not valid.

MQRC_LDAP_USER_NAME_ERROR
(2388, X'954') LDAP user name fields not valid.

MQRC_LDAP_USER_NAME_LENGTH_ERR
(2389, X'955') LDAP user name length not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_SCO_ERROR
(2380, X'94C') SSL configuration options structure not valid.

MQRC_SSL_CONFIG_ERROR
(2392, X'958') SSL configuration error.

For detailed information about these codes, see Reason codes.

Usage notes

For the Visual Basic programming language, the following point applies:

- The *ConnectOpts* parameter is declared as being of type MQCNO. If the application is running as a WebSphere MQ MQI client, and you want to specify the parameters of the client-connection channel, declare the *ConnectOpts* parameter as being of type Any, so that the application can specify an MQCNOCD structure on the call in place of an MQCNO structure. However, this means that the *ConnectOpts* parameter cannot be checked to ensure that it is the correct data type.

C invocation

```
MQCONNX (QMgrName, &ConnectOpts, &Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;     /* Name of queue manager */
MQCNO     ConnectOpts;  /* Options that control the action of MQCONNX */
MQHCONN   Hconn;        /* Connection handle */
MQLONG    CompCode;     /* Completion code */
MQLONG    Reason;       /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCONNX' USING QMGRNAME, CONNECTOPTS, HCONN, COMPCODE,
REASON.
```

Declare the parameters as follows:

```
**   Name of queue manager
 01  QMGRNAME     PIC X(48).
**   Options that control the action of MQCONNX
 01  CONNECTOPTS.
     COPY CMQCNOV.
**   Connection handle
 01  HCONN        PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE     PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON       PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCONNX (QMgrName, ConnectOpts, Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl QMgrName     char(48);       /* Name of queue manager */
dcl ConnectOpts  like MQCNO;     /* Options that control the action of
                                    MQCONNX */
dcl Hconn        fixed bin(31);  /* Connection handle */
dcl CompCode     fixed bin(31);  /* Completion code */
dcl Reason       fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQCONNX,(QMGRNAME,CONNECTOPTS,HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
QMGRNAME     DS       CL48  Name of queue manager
CONNECTOPTS  CMQCNOA  ,     Options that control the action of MQCONNX
HCONN        DS       F     Connection handle
COMPCODE     DS       F     Completion code
REASON       DS       F     Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQCONNX QMgrName, ConnectOpts, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim QMgrName    As String*48 'Name of queue manager'
Dim ConnectOpts As MQCNO     'Options that control the action of'
                             'MQCONNX'
Dim Hconn       As Long      'Connection handle'
Dim CompCode    As Long      'Completion code'
Dim Reason      As Long      'Reason code qualifying CompCode'
```

MQCRTMH - Create message handle:

The MQCRTMH call returns a message handle.

An application can use the MQCRTMH call on subsequent message queuing calls:

- Use the MQSETMP call to set a property of the message handle.
- Use the MQINQMP call to inquire on the value of a property of the message handle.
- Use the MQDLTMP call to delete a property of the message handle.

The message handle can be used on the MQPUT and MQPUT1 calls to associate the properties of the message handle with those of the message being put. Similarly by specifying a message handle on the MQGET call, the properties of the message being retrieved can be accessed using the message handle when the MQGET call completes.

Use MQDLTMH to delete the message handle.

Syntax

MQCRTMH (*Hconn*, *CrtMsgHOpts*, *Hmsg*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call. If the connection to the queue manager ceases to be valid and no WebSphere MQ call is operating on the message handle, MQDLTMH is implicitly called to delete the message.

Alternatively, you can specify the following value:

MQHC_UNASSOCIATED_HCONN

The connection handle does not represent a connection to any particular queue manager.

When this value is used, the message handle must be deleted with an explicit call to MQDLTMH in order to release any storage allocated to it; WebSphere MQ never implicitly deletes the message handle.

There must be at least one valid connection to a queue manager established on the thread creating the message handle, otherwise the call fails with MQRC_HCONN_ERROR.

In an environment with multiple installations on a single system, the MQHC_UNASSOCIATED_HCONN value is limited to use with the first installation loaded into the process. The reason code MQRC_HMSG_NOT_AVAILABLE is returned if the message handle is supplied to a different installation.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and you can specify the following value for *Hconn*:

MQHC_DEF_CONN

Default connection handle

CrtMsgHOpts

Type: MQCMHO - input

The options that control the action of MQCRTMH. See MQCMHO for details.

Hmsg

Type: MQHMSG - output

On output a message handle is returned that can be used to set, inquire, and delete properties of the message handle. Initially the message handle contains no properties.

A message handle also has an associated message descriptor. Initially this contains the default values. The values of the associated message descriptor fields can be set and inquired using the MQSETMP and MQINQMP calls. The MQDLTMP call resets a field of the message descriptor back to its default value.

If the *Hconn* parameter is specified as the value MQHC_UNASSOCIATED_HCONN then the returned message handle can be used on MQGET, MQPUT, or MQPUT1 calls with any connection within the unit of processing, but can only be in use by one WebSphere MQ call at a time. If the handle is in use when a second WebSphere MQ call attempts to use the same message handle, the second WebSphere MQ call fails with reason code MQRC_MSG_HANDLE_IN_USE.

If the *Hconn* parameter is not MQHC_UNASSOCIATED_HCONN then the returned message handle can only be used on the specified connection.

The same *Hconn* parameter value must be used on the subsequent MQI calls where this message handle is used:

- MQDLTMH
- MQSETMP
- MQINQMP
- MQDLTMP
- MQMHBUF
- MQBUFMH

The returned message handle ceases to be valid when the MQDLTMH call is issued for the message handle, or when the unit of processing that defines the scope of the handle terminates. MQDLTMH is called implicitly if a specific connection is supplied when the message handle is created and the connection to the queue manager ceases to be valid, for example, if MQDBC is called.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CMHO_ERROR
(2461, X'099D') Create message handle options structure not valid.

MQRC_CONNECTION_BROKEN
(2273, X'7D9') Connection to queue manager lost.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'07E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C

```
MQCRTMH (Hconn, &CrtMsgHOpts, &Hmsg, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;       /* Connection handle */
MQCMHO   CrtMsgHOpts; /* Options that control the action of MQCRTMH */
MQHMSG   Hmsg;        /* Message handle */
MQLONG   CompCode;    /* Completion code */
MQLONG   Reason;      /* Reason code qualifying CompCode */
```

COBOL

```
 CALL 'MQCRTMH' USING HCONN, CRTMSGOPTS, HMSG, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Options that control the action of MQCRTMH
 01  CRTMSGHOPTS.
     COPY CMQCMHOV.
**   Message handle
 01  HMSG    PIC S9(18) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I

```
call MQCRTMH (Hconn, CrtMsgHOpts, Hmsg, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31);  /* Connection handle */
dcl CrtMsgHOpts like MQCMHO;    /* Options that control the action of MQCRTMH */
dcl Hmsg        fixed bin(63);  /* Message handle */
dcl CompCode    fixed bin(31);  /* Completion code */
dcl Reason      fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler

```
CALL MQCRTMH,(HCONN,CRTMSGHOPTS,HMSG,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN          DS        F  Connection handle
CRTMSGHOPTS    CMQCMHOA  ,  Options that control the action of MQCRTMH
HMSG           DS        D  Message handle
COMPCODE       DS        F  Completion code
REASON         DS        F  Reason code qualifying COMPCODE
```

MQCTL - Control callbacks:

The MQCTL call performs controlling actions on callbacks and the object handles opened for a connection.

Syntax

MQCTL (*Hconn*, *Operation*,*ControlOpts*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and you can specify the following special value for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Operation

Type: MQLONG - input

The operation being processed on the callback defined for the specified object handle. You must specify one, and one only, of the following options:

MQOP_START

Start the consuming of messages for all defined message consumer functions for the specified connection handle.

Callbacks run on a thread started by the system, which is different from any of the application threads.

This operation gives control of the provided connection handle to system. The only MQI calls which can be issued by a thread other than the consumer thread are:

- MQCTL with Operation MQOP_STOP
- MQCTL with Operation MQOP_SUSPEND
- MQDISC - Performs MQCTL with Operation MQOP_STOP before disconnection the HConn.

MQRC_HCONN_ASYNC_ACTIVE is returned if a WebSphere MQ API call is issued while the connection handle is started, and the call does not originate from a message consumer function.

If a message consumer stops the connection during the MQCBCT_START_CALL then the MQCTL call returns with a failure reason code of MQRC_CONNECTION_STOPPED.

This can be issued in a consumer function. For the same connection as the callback routine, its only purpose is to cancel a previously issued MQOP_STOP operation.

This option is not supported in the following environments: CICS on z/OS or if the application is bound with a nonthreaded WebSphere MQ library.

MQOP_START_WAIT
Start the consuming of messages for all defined message consumer functions for the specified connection handle.

Message consumers run on the same thread and control is not returned to the caller of MQCTL until:

- Released by the use of the MQCTL MQOP_STOP or MQOP_SUSPEND operations, or
- All consumer routines have been deregistered or suspended.

If all consumers are deregistered or suspended, an implicit MQOP_STOP operation is issued.

This option cannot be used from within a callback routine, either for the current connection handle or any other connection handle. If the call is attempted it returns with MQRC_ENVIRONMENT_ERROR.

If, at any time during an MQOP_START_WAIT operation there are no registered, non-suspended consumers the call fails with a reason code of MQRC_NO_CALLBACKS_ACTIVE.

If, during an MQOP_START_WAIT operation, the connection is suspended, the MQCTL call returns a warning reason code of MQRC_CONNECTION_SUSPENDED; the connection remains 'started'.

The application can choose to issue MQOP_STOP or MQOP_RESUME. In this instance, the MQOP_RESUME operation blocks.

This option is not supported in a single threaded client.

MQOP_STOP
Stop the consuming of messages, and wait for all consumers to complete their operations before this option completes. This operation releases the connection handle.

If issued from within a callback routine, this option does not take effect until the routine exits. No more message consumer routines are called after the consumer routines for messages already read have completed, and after stop calls (if requested) to callback routines have been made.

If issued outside a callback routine, control does not return to the caller until the consumer routines for messages already read have completed, and after stop calls (if requested) to callbacks have been made. The callbacks themselves, however, remain registered.

This function has no effect on read ahead messages. You must ensure that consumers run MQCLOSE(MQCO_QUIESCE), from within the callback function, to determine whether there are any further messages available to be delivered.

MQOP_SUSPEND
Pause the consuming of messages. This operation releases the connection handle.

This does not have any effect on the reading ahead of messages for the application. If you intend to stop consuming messages for a long time, consider closing the queue and reopening it when consumption continues.

If issued from within a callback routine, it does not take effect until the routine exits. No more message consumer routines will be called after the current routine exits.

If issued outside a callback, control does not return to the caller until the current consumer routine has completed and no more are called.

MQOP_RESUME
Resume the consuming of messages.

This option is normally issued from the main application thread, but it can also be used from within a callback routine to cancel an earlier suspension request issued in the same routine.

If the MQOP_RESUME is used to resume an MQOP_START_WAIT then the operation blocks.

ControlOpts
Type: MQCTLO - input

Options that control the action of MQCTL

See MQCTLO for details of the structure.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'855') Unable to load data conversion services modules.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALLBACK_LINK_ERROR
(2487, X'9B7') Unable to call the callback routine

MQRC_CALLBACK_NOT_ REGISTERED
(2448, X'990') Unable to Deregister, Suspend, or Resume because there is no registered callback

MQRC_CALLBACK_ROUTINE_ERROR
(2486, X'9B6') Either, both CallbackFunction and CallbackName have been specified on an MQOP_REGISTER call.

Or either CallbackFunction or CallbackName have been specified but does not match the currently registered callback function.

MQRC_CALLBACK_TYPE_ERROR
(2483, X'9B3') Incorrect CallBackType field.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CBD_ERROR
(2444, X'98C') Option block is incorrect.

MQRC_CBD_OPTIONS_ERROR
(2484, X'9B4') Incorrect MQCBD options field.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CORREL_ID_ERROR
(2207, X'89F') Correlation-identifier error.

MQRC_FUNCTION_NOT_SUPPORTED
(2298, X'8FA') The function requested is not available in the current environment.

MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_INVALID_MSG_UNDER_CURSOR
(2246, X'8C6') Message under cursor not valid for retrieval.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MATCH_OPTIONS_ERROR
(2247, X'8C7') Match options not valid.

MQRC_MAX_MSG_LENGTH_ERROR
(2485, X'9B5') Incorrect MaxMsgLength field

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MODULE_ENTRY_NOT_FOUND
(2497, X'9C1')The specified function entry point could not be found in the module.

MQRC_MODULE_INVALID
(2496, X'9C0') Module is found but is of the wrong type (32 bit/64 bit) or is not a valid dll.

MQRC_MODULE_NOT_FOUND
(2495, X'9BF') Module not found in the search path or not authorized to load.

MQRC_MSG_ID_ERROR
(2206, X'89E') Message-identifier error.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Use of message token not valid.

MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'7F4') Queue not open for browse.

MQRC_NOT_OPEN_FOR_INPUT
(2037, X'7F5') Queue not open for input.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPERATION_ERROR
(2488, X'9B8') Incorrect Operation code on API Call

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_INDEX_TYPE_ERROR
(2394, X'95A') Queue has wrong index type.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SIGNAL_OUTSTANDING
(2069, X'815') Signal outstanding for this handle.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Syncpoint support not available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION
(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

For detailed information about these codes, see Reason codes.

Usage notes

1. Callback routines must check the responses from all services they invoke, and if the routine detects a condition that cannot be resolved, it must issue an MQCB MQOP_DEREGISTER command to prevent repeated calls to the callback routine.
2. On z/OS, when Operation is MQOP_START:
   - Programs which use asynchronous callback routines must be authorized to use z/OS UNIX System Services (USS).
   - Language Environment (LE) programs which use asynchronous callback routines must use the LE runtime option POSIX(ON).
   - Non-LE programs which use asynchronous callback routines must not use the USS pthread_create interface (callable service BPX1PTC).
3. MQCTL is not supported within the IMS adapter.

Note: In CICS, MQOP_START is not supported. Instead, use the MQOP_START_WAIT function call.

C invocation

```
MQCTL (Hconn, Operation, &ControlOpts, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQLONG   Operation;     /* Operation being processed */
MQCTLO   ControlOpts    /* Options that control the action of MQCTL */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQCTL' USING HCONN, OPERATION, CTLOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Operation
 01  OPERATION PIC S9(9) BINARY.
**   Control Options
 01  CTLOPTS.
     COPY CMQCTLOV.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQCTL(Hconn, Operation, CtlOpts, CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn        fixed bin(31); /* Connection handle */
dcl Operation    fixed bin(31); /* Operation */
dcl CtlOpts like MQCTLO;        /* Options that control the action of MQCTL */
dcl CompCode     fixed bin(31); /* Completion code */
dcl Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

MQDISC - Disconnect queue manager:

The MQDISC call breaks the connection between the queue manager and the application program, and is the inverse of the MQCONN or MQCONNX call.

- On z/OS, all applications that use asynchronous message consumption, event handling or callback, the main control thread must issue an MQDISC call before ending. See Asynchronous consumption of WebSphere MQ messages for more details.
- On z/OS, CICS applications do not need to issue this call to disconnect from the queue manager, but might need to issue it to end the use of a connection tag.
- On IBM i, applications running in compatibility mode do not need to issue this call. See "MQCONN - Connect queue manager" on page 1961 for more information.

Syntax

MQDISC (*Hconn*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input/output

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, you can omit the MQCONN call, and specify the following value for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

On successful completion of the call, the queue manager sets *Hconn* to a value that is not a valid handle for the environment. This value is:

MQHC_UNUSABLE_HCONN
Unusable connection handle.

On z/OS, *Hconn* is set to a value that is undefined.

CompCode
Type: MQLONG - output

The completion code; it is one of the following codes:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_CONN_TAG_NOT_RELEASED
(2344, X'928') Connection tag not released.

MQRC_OUTCOME_PENDING
(2124, X'84C') Result of commit operation is pending.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_DISC_LOAD_ERROR
(2138, X'85A') Unable to load adapter disconnection module.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_INIT_ERROR
(2375, X'947') API exit initialization failed.

MQRC_API_EXIT_TERM_ERROR
(2376, X'948') API exit termination failed.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OUTCOME_MIXED
(2123, X'84B') Result of commit or back-out operation is mixed.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. If an MQDISC call is issued when the connection still has objects open under that connection, the queue manager closes those objects, with the close options set to MQCO_NONE.
2. If the application ends with uncommitted changes in a unit of work, the disposition of those changes depends on how the application ends:
   a. If the application issues the MQDISC call before ending:
      - For a queue-manager-coordinated unit of work, the queue manager issues the MQCMIT call on behalf of the application. The unit of work is committed if possible, and backed out if not.
      - For an externally coordinated unit of work, there is no change in the status of the unit of work; however, the queue manager typically indicates that the unit of work must be committed when asked by the unit-of-work coordinator.

        On z/OS, CICS, IMS (other than batch DL/1 programs), and RRS applications are like this.

   b. If the application ends normally but without issuing the MQDISC call, the action taken depends on the environment:
      - On z/OS, except for MQ Java or MQ JMS applications, the actions described in note 2a occur.
      - In all other cases, the actions described in note 2c occur.

      Because of the differences between environments, ensure that applications that you want to port either commit or back out the unit of work before they end.
   c. If the application ends *abnormally* without issuing the MQDISC call, the unit of work is backed out.
3. On z/OS, the following points apply:

- CICS applications do not have to issue the MQDISC call to disconnect from the queue manager, because the CICS system itself connects to the queue manager, and the MQDISC call has no effect on this connection.
- CICS, IMS (other than batch DL/1 programs), and RRS applications use units of work that are coordinated by an external unit-of-work coordinator. As a result, the MQDISC call does not affect the status of the unit of work (if any) that exists when the call is issued.

  However the MQDISC call *does* indicate the end of use of the connection tag *ConnTag* that was associated with the connection by an earlier MQCONNX call issued by the application. If there is an active unit of work that references the connection tag when the MQDISC call is issued, the call completes with completion code MQCC_WARNING and reason code MQRC_CONN_TAG_NOT_RELEASED. The connection tag does not become available for reuse until the external unit-of-work coordinator has resolved the unit of work.

4. On IBM i, applications running in compatibility mode do not have to issue this call; see the MQCONN call for more details.

Note: In CICS, MQOP_START is not supported. Instead, use the MQOP_START_WAIT function call.

C invocation

```
MQDISC (&Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDISC' USING HCONN, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDISC (Hconn, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
CALL MQDISC,(HCONN,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS  F  Connection handle
COMPCODE  DS  F  Completion code
REASON    DS  F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQDISC Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQDLTMH - Delete message handle:

The MQDLTMH call deletes a message handle and is the inverse of the MQCRTMH call.

Syntax

MQDLTMH (*Hconn*, *Hmsg*, *DltMsgHOpts*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager.

The value must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread deleting the message handle, otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMSG - input/output

This is the message handle to be deleted. The value was returned by a previous MQCRTMH call.

On successful completion of the call, the handle is set to an invalid value for the environment. This value is:

MQHM_UNUSABLE_HMSG
Unusable message handle.

The message handle cannot be deleted if another WebSphere MQ call is in progress that was passed the same message handle.

DltMsgHOpts
Type: MQDMHO - input

See MQDMHO for details.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DMHO_ERROR
(2462, X'099E') Delete message handle options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQDLTMH (Hconn, &Hmsg, &DltMsgHOpts, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;       /* Connection handle */
MQHMSG   Hmsg;        /* Message handle */
MQDMHO   DltMsgHOpts; /* Options that control the action of MQDLTMH */
MQLONG   CompCode;    /* Completion code */
MQLONG   Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDLTMH' USING HCONN, HMSG, DLTMSGOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01  HCONN  PIC S9(9) BINARY.

** Options that control the action of MQDLTMH
01  DLTMSGHOPTS.
   COPY CMQDLMHOV.

** Completion code
01 COMPCODE  PIC S9(9) BINARY.

** Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDLTMH (Hconn, Hmsg, DltMsgHOpts, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31);  /* Connection handle */
dcl Hmsg        fixed bin(63);  /* Message handle */
dcl DltMsgHOpts like MQDMHO;    /* Options that control the action of MQDLTMH */
dcl CompCode    fixed bin(31);  /* Completion code */
dcl Reason      fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQDLTMH,(HCONN,HMSG,DLTMSGHOPTS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN          DS        F  Connection handle
HMSG           DS        D  Message handle
DLTMSGHOPTS    CMQDMHOA  ,  Options that control the action of MQDLTMH
COMPCODE       DS        F  Completion code
REASON         DS        F  Reason code qualifying COMPCODE
```

MQDLTMP - Delete message property:

The MQDLTMP call deletes a property from a message handle and is the inverse of the MQSETMP call.

Syntax

MQDLTMP (*Hconn*, *Hmsg*, *DltPropOpts*, *Name*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread deleting the message handle otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg

Type: MQHMSG - input

This is the message handle containing the property to be deleted. The value was returned by a previous MQCRTMH call.

DltPropOpts

Type: MQDMPO - input

See the MQDMPO data type for details.

Name

Type: MQCHARV - input

The name of the property to delete. See Property names for further information about property names.

Wildcards are not allowed in the property name.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7') Property not available.

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'0852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'086D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DMPO_ERROR
(2481, X'09B1') Delete message property options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'0893') Unexpected error occurred.

For detailed information about these codes, see:

- Reason codes for WebSphere MQ for z/OS
- API reason codes for other WebSphere MQ platforms

.

C invocation

```
MQDLTMP (Hconn, Hmsg, &DltPropOpts, &Name, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn;       /* Connection handle */
MQHMSG  Hmsg;        /* Message handle */
MQDMPO  DltPropOpts; /* Options that control the action of MQDLTMP */
MQCHARV Name;        /* Property name */
MQLONG  CompCode;    /* Completion code */
MQLONG  Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQDLTMP' USING HCONN, HMSG, DLTPROPOPTS, NAME, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Message handle
 01  HMSG   PIC S9(18) BINARY.
**   Options that control the action of MQDLTMP
 01  DLTPROPOPTS.
     COPY CMQDMPOV.
**   Property name
 01  NAME
     COPY CMQCHRVV.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQDLTMP (Hconn, Hmsg, DltPropOpts, Name, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31); /* Connection handle */
dcl Hmsg        fixed bin(63); /* Message handle */
dcl DltPropOpts like MQDMPO;   /* Options that control the action of MQDLTMP */
dcl Name        like MQCHARV;  /* Property name */
dcl CompCode    fixed bin(31); /* Completion code */
dcl Reason      fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQDLTMP,(HCONN,HMSG,DLTPROPOPTS,NAME,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS         F     Connection handle
HMSG         DS         D     Message handle
DLTPROPOPTS  CMQDMPOA   ,     Options that control the action of MQDLTMP
NAME         CMQCHRVA   ,     Property name
COMPCODE     DS         F     Completion code
REASON       DS         F     Reason code qualifying COMPCODE
```

MQGET - Get message:

The MQGET call retrieves a message from a local queue that has been opened using the MQOPEN call.

Syntax

MQGET (*Hconn*, *Hobj*, *MsgDesc*, *GetMsgOpts*, *BufferLength*, *Buffer*, *DataLength*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj

Type: MQHOBJ - input

This handle represents the queue from which a message is to be retrieved. The value of *Hobj* was returned by a previous MQOPEN call. The queue must have been opened with one or more of the following options (see "MQOPEN - Open object" on page 2030 for details):

- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE
- MQOO_INPUT_AS_Q_DEF
- MQOO_BROWSE

MsgDesc

Type: MQMD - input/output

This structure describes the attributes of the message required, and the attributes of the message retrieved. See "MQMD - Message descriptor" on page 1705 for details.

If *BufferLength* is less than the message length, *MsgDesc* is filled by the queue manager, whether MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see MQGMO - Options field).

If the application provides a version-1 MQMD, the message returned has an MQMDE prefixed to the application message data, but *only* if one or more of the fields in the MQMDE has a nondefault value. If all the fields in the MQMDE have default values, the MQMDE is omitted. A format name of MQFMT_MD_EXTENSION in the *Format* field in MQMD indicates that an MQMDE is present.

The application does not need to provide an MQMD structure if a valid message handle is supplied in the *MsgHandle* field. If nothing is provided in this field, the descriptor of the message is taken from the descriptor associated with the message handles.

If the application provides a message handle rather than an MQMD structure, and specifies MQGMO_PROPERTIES_FORCE_MQRFH2, the call fails with reason code MQRC_MD_ERROR. The call also fails, with reason code MQRC_MD_ERROR, if the application does not provide an MQMD structure and specifies MQGMO_PROPERTIES_AS_Q_DEF, and the *PropertyControl* queue attribute is MQPROP_FORCE_MQRFH2.

If match options are specified and the message descriptor associated with the message handle is being used, the input fields used for matching come from the message handle.

GetMsgOpts
Type: MQGMO - input/output

See “MQGMO - Get-message options” on page 1655 for details.

BufferLength
Type: MQLONG - input

This is the length in bytes of the *Buffer* area. Specify zero for messages that have no data, or if the message is to be removed from the queue and the data discarded (you must specify MQGMO_ACCEPT_TRUNCATED_MSG in this case).

Note: The length of the longest message that it is possible to read from the queue is given by the *MaxMsgLength* queue attribute; see “Attributes for queues” on page 2135.

Buffer
Type: MQBYTExBufferLength - output

This is the area to contain the message data. Align the buffer on a boundary appropriate to the nature of the data in the message. 4 byte alignment is suitable for most messages (including messages containing WebSphere MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64 bit binary integer might require 8-byte alignment.

If *BufferLength* is less than the message length, as much of the message as possible is moved into *Buffer*; this happens whether MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see MQGMO - Options field for more information).

The character set and encoding of the data in *Buffer* are given by the *CodedCharSetId* and *Encoding* fields returned in the *MsgDesc* parameter. If these values are different from the values required by the receiver, the receiver must convert the application message data to the character set and encoding required. The MQGMO_CONVERT option can be used (with a user-written exit if necessary) to convert the message data; see “MQGMO - Get-message options” on page 1655 for details of this option.

Note: All the other parameters on the MQGET call are in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE).

If the call fails, the contents of the buffer might still have changed.

In the C programming language, the parameter is declared as a pointer-to-void: the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

DataLength
Type: MQLONG - output

This is the length in bytes of the application data *in the message*. If this value is greater than *BufferLength*, only *BufferLength* bytes are returned in the *Buffer* parameter (that is, the message is truncated). If the value is zero, the message contains no application data.

If *BufferLength* is less than the message length, *DataLength* is still completed by the queue manager, whether MQGMO_ACCEPT_TRUNCATED_MSG is specified on the *GetMsgOpts* parameter (see MQGMO - Options field for more information). This allows the application to determine the size of the buffer required to accommodate the message data, and then reissue the call with a buffer of the appropriate size.

However, if the MQGMO_CONVERT option is specified, and the converted message data is too long to fit in *Buffer*, the value returned for *DataLength* is:

- The length of the *unconverted* data, for queue-manager defined formats.

In this case, if the nature of the data causes it to expand during conversion, the application must allocate a buffer bigger than the value returned by the queue manager for *DataLength*.

- The value returned by the data-conversion exit, for application-defined formats.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason codes listed are the ones that the queue manager can return for the *Reason* parameter. If the application specifies the MQGMO_CONVERT option, and a user-written exit is invoked to convert some or all the message data, the exit decides what value is returned for the *Reason* parameter. As a result, values other than those values documented are possible.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too large for buffer.

MQRC_CONVERTED_STRING_TOO_BIG
(2190, X'88E') Converted string too large for field.

MQRC_DBCS_ERROR
(2150, X'866') DBCS string not valid.

MQRC_FORMAT_ERROR
(2110, X'83E') Message format not valid.

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_INCONSISTENT_CCSIDS
(2243, X'8C3') Message segments have differing CCSIDs.

MQRC_INCONSISTENT_ENCODINGS
(2244, X'8C4') Message segments have differing encodings.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Invalid use of message token.

MQRC_NO_MSG_LOCKED
(2209, X'8A1') No message locked.

MQRC_NOT_CONVERTED
(2119, X'847') Message data not converted.

MQRC_OPTIONS_CHANGED
(nnnn, X'xxx') Options that were required to be consistent have been changed.

MQRC_PARTIALLY_CONVERTED
(2272, X'8E0') Message data partially converted.

MQRC_SIGNAL_REQUEST_ACCEPTED
(2070, X'816') No message returned (but signal request accepted).

MQRC_SOURCE_BUFFER_ERROR
(2145, X'861') Source buffer parameter not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR
(2113, X'841') Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR
(2114, X'842') Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_SOURCE_LENGTH_ERROR
(2143, X'85F') Source length parameter not valid.

MQRC_TARGET_BUFFER_ERROR
(2146, X'862') Target buffer parameter not valid.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'846') Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).

MQRC_TRUNCATED_MSG_FAILED
(2080, X'820') Truncated message returned (processing not completed).

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_CONV_LOAD_ERROR
(2133, X'855') Unable to load data conversion services modules.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CORREL_ID_ERROR
(2207, X'89F') Correlation-identifier error.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') Db2 subsystem not available.

MQRC_GET_INHIBITED
(2016, X'7E0') Gets inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GMO_ERROR
(2186, X'88A') Get-message options structure not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INCONSISTENT_BROWSE
(2259, X'8D3') Inconsistent browse specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_INVALID_MSG_UNDER_CURSOR
(2246, X'8C6') Message under cursor not valid for retrieval.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MATCH_OPTIONS_ERROR
(2247, X'8C7') Match options not valid.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MSG_ID_ERROR
(2206, X'89E') Message-identifier error.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOKEN_ERROR
(2331, X'91B') Use of message token not valid.

MQRC_NO_MSG_AVAILABLE
(2033, X'7F1') No message available.

MQRC_NO_MSG_UNDER_CURSOR
(2034, X'7F2') Browse cursor not positioned on message.

MQRC_NOT_OPEN_FOR_BROWSE
(2036, X'7F4') Queue not open for browse.

MQRC_NOT_OPEN_FOR_INPUT
(2037, X'7F5') Queue not open for input.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_INDEX_TYPE_ERROR
(2394, X'95A') Queue has wrong index type.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SECOND_MARK_NOT_ALLOWED
(2062, X'80E') A message is already marked.

MQRC_SIGNAL_OUTSTANDING
(2069, X'815') Signal outstanding for this handle.

MQRC_SIGNAL1_ERROR
(2099, X'833') Signal field not valid.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') sync point support not available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
(2090, X'82A') Wait interval in MQGMO not valid.

MQRC_WRONG_GMO_VERSION
(2256, X'8D0') Wrong version of MQGMO supplied.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

For detailed information about these codes, see Reason codes.

Usage notes

1. The message retrieved is normally deleted from the queue. This deletion can occur as part of the MQGET call itself, or as part of a sync point.

   The browse options are: MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, and MQGMO_BROWSE_MSG_UNDER_CURSOR.
2. If the MQGMO_LOCK option is specified with one of the browse options, the browsed message is locked so that it is visible only to this handle.

   If the MQGMO_UNLOCK option is specified, a previously locked message is unlocked. No message is retrieved in this case, and the *MsgDesc*, *BufferLength*, *Buffer*, and *DataLength* parameters are not checked or altered.

3. For applications issuing an MQGET call, the message retrieved can be lost if the application terminates abnormally or the connection is severed while processing the call. This issue arises because the surrogate running on the same platform as the queue manager that issues the MQGET call on behalf of the application cannot detect the loss of the application until the surrogate is about to return the message to the application, *after* the message has been removed from the queue. This issue can occur for both persistent messages and nonpersistent messages.

   To eliminate the risk of losing messages in this way, always retrieve messages within units of work. That is, by specifying the MQGMO_SYNCPOINT option on the MQGET call, and using the MQCMIT or MQBACK calls to commit or back out the unit of work when message processing is complete. If MQGMO_SYNCPOINT is specified, and the client terminates abnormally or the connection is severed, the surrogate backs out the unit of work on the queue manager and the message is reinstated on the queue. For more information about sync points, see Syncpoint considerations in WebSphere MQ applications.

   This situation can arise with WebSphere MQ clients as well as with applications that are running on the same platform as the queue-manager.
4. If an application puts a sequence of messages on a particular queue within a single unit of work, and then commits that unit of work successfully, the messages become available for retrieval as follows:
   - If the queue is a *nonshared* queue (that is, a local queue), all messages within the unit of work become available at the same time.
   - If the queue is a *shared* queue, messages within the unit of work become available in the order in which they were put, but not all at the same time. When the system is heavily laden, it is possible for the first message in the unit of work to be retrieved successfully, but for the MQGET call for the second or subsequent message in the unit of work to fail with MQRC_NO_MSG_AVAILABLE. If this issue occurs, the application must wait a short while and then try the operation again.
5. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if certain conditions are satisfied. See MQPUT usage notes for details. If the conditions are satisfied, the messages are presented to the receiving application in the order in which they were sent, if:
   - Only one receiver is getting messages from the queue.

     If there are two or more applications getting messages from the queue, they must agree with the sender the mechanism to be used to identify messages that belong to a sequence. For example, the sender might set all the *CorrelId* fields in the messages in a sequence to a value that was unique to that sequence of messages.
   - The receiver does not deliberately change the order of retrieval, for example by specifying a particular *MsgId* or *CorrelId*.

   If the sending application puts the messages as a message group, the messages are presented to the receiving application in the correct order if the receiving application specifies the MQGMO_LOGICAL_ORDER option on the MQGET call. For more information about message groups, see:
   - MQMD - MsgFlags field
   - MQPMO_LOGICAL_ORDER
   - MQGMO_LOGICAL_ORDER

   If the user is getting messages in a group under sync point, they must ensure that the complete group is processed before attempting to finish the transaction.
6. Applications must test for the feedback code MQFB_QUIT in the *Feedback* field of the *MsgDesc* parameter, and end if they find this value. See MQMD - Feedback field for more information.
7. If the queue identified by *Hobj* was opened with the MQOO_SAVE_ALL_CONTEXT option, and the completion code from the MQGET call is MQCC_OK or MQCC_WARNING, the context associated with the queue handle *Hobj* is set to the context of the message that has been retrieved (unless the

MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option is set, in which case the context is marked as not available).

You can use the saved context on a subsequent MQPUT or MQPUT1 call by specifying the MQPMO_PASS_IDENTITY_CONTEXT or MQPMO_PASS_ALL_CONTEXT options. This enables the context of the message received to be transferred in whole or in part to another message (for example, when the message is forwarded to another queue). For more information about message context, see Message context.

8. If you include the MQGMO_CONVERT option in the *GetMsgOpts* parameter, the application message data is converted to the representation requested by the receiving application, before the data is placed in the *Buffer* parameter:
   - The *Format* field in the control information in the message identifies the structure of the application data, and the *CodedCharSetId* and *Encoding* fields in the control information in the message specify its character-set identifier and encoding.
   - The application issuing the MQGET call specifies in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter the character-set identifier and encoding to which to convert the application message data.

   When conversion of the message data is necessary, the conversion is performed either by the queue manager itself or by a user-written exit, depending on the value of the *Format* field in the control information in the message:
   - The following format names are formats that are converted by the queue manager; these formats are called "built-in" formats:
     - MQFMT_ADMIN
     - MQFMT_CICS (z/OS only)
     - MQFMT_COMMAND_1
     - MQFMT_COMMAND_2
     - MQFMT_DEAD_LETTER_HEADER
     - MQFMT_DIST_HEADER
     - MQFMT_EVENT version 1
     - MQFMT_EVENT version 2 (z/OS only)
     - MQFMT_IMS
     - MQFMT_IMS_VAR_STRING
     - MQFMT_MD_EXTENSION
     - MQFMT_PCF
     - MQFMT_REF_MSG_HEADER
     - MQFMT_RF_HEADER
     - MQFMT_RF_HEADER_2
     - MQFMT_STRING
     - MQFMT_TRIGGER
     - MQFMT_WORK_INFO_HEADER (z/OS only)
     - MQFMT_XMIT_Q_HEADER
   - The format name MQFMT_NONE is a special value that indicates that the nature of the data in the message is undefined. As a consequence, the queue manager does not attempt conversion when the message is retrieved from the queue.

     Note: If MQGMO_CONVERT is specified on the MQGET call for a message that has a format name of MQFMT_NONE, and the character set or encoding of the message differs from that

specified in the *MsgDesc* parameter, the message is returned in the *Buffer* parameter (assuming no other errors), but the call completes with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

You can use MQFMT_NONE either when the nature of the message data means that it does not require conversion, or when the sending and receiving applications have agreed between themselves the form in which to send the message data.

- All other format names pass the message to a user-written exit for conversion. The exit has the same name as the format, apart from environment-specific additions. User-specified format names must not begin with the letters WebSphere MQ.

  See "Data conversion" on page 2210 for details of the data-conversion exit.

User data in the message can be converted between any supported character sets and encodings. However, be aware that, if the message contains one or more WebSphere MQ header structures, the message cannot be converted from or to a character set that has double-byte or multi-byte characters for any of the characters that are valid in queue names. Reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR results if this is attempted, and the message is returned unconverted. Unicode character set UCS-2 is an example of such a character set.

On return from MQGET, the following reason code indicates that the message was converted successfully:

- MQRC_NONE

The following reason code indicates that the message *might* have been converted successfully; the application must check the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to find out:

- MQRC_TRUNCATED_MSG_ACCEPTED

All other reason codes indicate that the message was not converted.

Note: The interpretation of this reason code is true for conversions performed by a user-written exit *only* if the exit conforms to the processing guidelines described in "Data conversion" on page 2210.

9. When using the object-oriented interface to get messages, you can choose not to specify a buffer to hold the message data for an MQGET call. However, in previous versions of WebSphere MQ, it was possible for MQGET to fail with reason code MQRC_CONVERTED_MSG_TO_BIG, even when a buffer was not specified. In WebSphere MQ Version 7, when you get a message using an object-oriented application without restricting the size of the receive message buffer, the application does not fail with MQRC_CONVERTED_MSG_TOO_BIG, and receives the converted message. This is true of the following environments:
   - .NET, including fully managed applications
   - C++
   - Java (WebSphere MQ classes for Java)

   Note: For all clients, if the value of *sharingConversations* is zero, the channel operates as it did before WebSphere MQ Version 7.0, and message handling reverts to Version 6 behavior. In this situation, if the buffer is too small to receive the converted message, the unconverted message is returned, with reason code MQRC_CONVERTED_MSG_TOO_BIG. For more information about *sharingConversations*, see Using sharing conversations in a client application.
10. For the built-in formats, the queue manager can perform *default conversion* of character strings in the message when the MQGMO_CONVERT option is specified. Default conversion allows the queue manager to use an installation-specified default character set that approximates the actual character set, when converting string data. As a result, the MQGET call can succeed with completion code MQCC_OK, instead of completing with MQCC_WARNING and reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR.

Note: The result of using an approximate character set to convert string data is that some characters might be converted incorrectly. To avoid this, use characters in the string that are common to both the actual character set and the default character set.

Default conversion applies both to the application message data and to character fields in the MQMD and MQMDE structures:

- Default conversion of the application message data occurs only when *all* the following are true:
  - The application specifies MQGMO_CONVERT.
  - The message contains data that must be converted either from or to a character set that is not supported.
  - Default conversion was enabled when the queue manager was installed or restarted.
- Default conversion of the character fields in the MQMD and MQMDE structures occurs as necessary, if default conversion is enabled for the queue manager. The conversion is performed even if the MQGMO_CONVERT option is not specified by the application on the MQGET call.

11. For the Visual Basic programming language, the following points apply:
    - If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_STORAGE_NOT_AVAILABLE.
    - The *Buffer* parameter is declared as being of type `String`. If the data to be retrieved from the queue is not of type `String`, use the MQGETAny call in place of MQGET.

      The MQGETAny call has the same parameters as the MQGET call, except that the *Buffer* parameter is declared as being of type `Any`, allowing any type of data to be retrieved. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.
12. Not all MQGET options are supported when read ahead is enabled. The following table indicated which options are allowed and whether they can be altered between MQGET calls.

Table 213. MQGET options permitted when read ahead is enabled

	Permitted when read ahead is enabled and can be altered between MQGET calls	Permitted when read ahead is enabled but cannot be altered between MQGET calls[a]	MQGET options that are not permitted when read ahead is enabled[b]
MQGET MD values	MsgId[c] CorrelId[c]	Encoding CodedCharSetId	
MQGET MQGMO options	MQGMO_WAIT MQGMO_NO_WAIT MQGMO_FAIL_IF_QUIESCING MQGMO_BROWSE_FIRST[d] MQGMO_BROWSE_NEXT[d] MQGMO_BROWSE_MESSAGE _UNDER_CURSOR[d]	MQGMO_SYNCPOINT_IF_PERSISTENT MQGMO_NO_SYNCPOINT MQGMO_ACCEPT_TRUNCATED_MSG MQGMO_CONVERT MQGMO_LOGICAL_ORDER MQGMO_COMPLETE_MSG MQGMO_ALL_MSGS_AVAILABLE MQGMO_ALL_SEGMENTS_AVAILABLE MQGMO_MARK_BROWSE_HANDLE MQGMO_MARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_CO_OP MQGMO_UNMARK_BROWSE_HANDLE MQGMO_UNMARKED_BROWSE_MSG MQGMO_PROPERTIES_FORCE_MQRFH2 MQGMO_NO_PROPERTIES MQGMO_PROPERTIES_IN_HANDLE MQGMO_PROPERTIES_COMPATIBILITY	MQGMO_SET_SIGNAL MQGMO_SYNCPOINT MQGMO_MARK_SKIP _BACKOUT MQGMO_MSG_UNDER _CURSOR[d] MQGMO_LOCK MQGMO_UNLOCK
MQGMO values		MsgHandle	

a. If these options are altered between MQGET calls an MQRC_OPTIONS_CHANGED reason code is returned.

b. If these options are specified on the first MQGET call then read ahead is disabled. If these options are specified on a subsequent MQGET call a reason code MQRC_OPTIONS_ERROR is returned.

c. The client applications need to be aware that if the MsgId and CorrelId values are altered between MQGET calls messages with the previous values might have already been sent to the client and remain in the client read ahead buffer until consumed (or automatically purged).

d. The first MQGET call determines whether messages are to be browsed or got from a queue when read ahead is enabled. If the application attempts to use a combination of browse and get an MQRC_OPTIONS_CHANGED reason code is returned.

e. MQGMO_MSG_UNDER_CURSOR is not possible with read ahead. Messages can be browsed or got when read ahead is enabled but not a combination of both.

13. Applications can destructively get uncommitted messages only if those messages are put in the same local unit of work as the get. Applications cannot get uncommitted messages nondestructively.
14. Messages under a browse cursor can be retrieved in a unit of work. It is not possible to retrieve an uncommitted message in this way.

C invocation

```
MQGET (Hconn, Hobj, &MsgDesc, &GetMsgOpts, BufferLength, Buffer,
      &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQHOBJ   Hobj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQGMO    GetMsgOpts;    /* Options that control the action of MQGET */
MQLONG   BufferLength;  /* Length in bytes of the Buffer area */
MQBYTE   Buffer[n];     /* Area to contain the message data */
MQLONG   DataLength;    /* Length of the message */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQGET' USING HCONN, HOBJ, MSGDESC, GETMSGOPTS, BUFFERLENGTH,
BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Object handle
 01  HOBJ          PIC S9(9) BINARY.
**   Message descriptor
 01  MSGDESC.
     COPY CMQMDV.
**   Options that control the action of MQGET
 01  GETMSGOPTS.
     COPY CMQGMOV.
**   Length in bytes of the BUFFER area
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Area to contain the message data
 01  BUFFER        PIC X(n).
**   Length of the message
 01  DATALENGTH    PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
           DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl Hobj          fixed bin(31);  /* Object handle */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl GetMsgOpts    like MQGMO;     /* Options that control the action of
                                     MQGET */
dcl BufferLength  fixed bin(31);  /* Length in bytes of the Buffer
                                     area */
dcl Buffer        char(n);        /* Area to contain the message data */
```

```
dcl DataLength    fixed bin(31);  /* Length of the message */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
        CALL MQGET,(HCONN,HOBJ,MSGDESC,GETMSGOPTS,BUFFERLENGTH,
             BUFFER,DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN         DS       F      Connection handle
HOBJ          DS       F      Object handle
MSGDESC       CMQMDA   ,      Message descriptor
GETMSGOPTS    CMQGMOA  ,      Options that control the action of MQGET
BUFFERLENGTH  DS       F      Length in bytes of the BUFFER area
BUFFER        DS       CL(n)  Area to contain the message data
DATALENGTH    DS       F      Length of the message
COMPCODE      DS       F      Completion code
REASON        DS       F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQGET Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
DataLength, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim Hobj         As Long   'Object handle'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim GetMsgOpts   As MQGMO  'Options that control the action of MQGET'
Dim BufferLength As Long   'Length in bytes of the Buffer area'
Dim Buffer       As String 'Area to contain the message data'
Dim DataLength   As Long   'Length of the message'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

MQINQ - Inquire object attributes:

The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

The following types of object are valid:

- Queue manager
- Queue
- Namelist
- Process definition

Syntax

MQINQ (*Hconn*, *Hobj*, *SelectorCount*, *Selectors*, *IntAttrCount*, *IntAttrs*, *CharAttrLength*, *CharAttrs*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj
Type: MQHOBJ - input

This handle represents the object (of any type) with attributes that are required. The handle must be returned by a previous MQOPEN call that specified the MQOO_INQUIRE option.

SelectorCount
Type: MQLONG - input

This is the count of selectors that are supplied in the *Selectors* array. It is the number of attributes that are to be returned. Zero is a valid value. The maximum number allowed is 256.

Selectors
Type: MQLONG × *SelectorCount* - input

This is an array of *SelectorCount* attribute selectors; each selector identifies an attribute (integer or character) with a value that is required.

Each selector must be valid for the type of object that *Hobj* represents, otherwise the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.

In the special case of queues:

- If the selector is not valid for queues of any type, the call fails with completion code MQCC_FAILED and reason code MQRC_SELECTOR_ERROR.
- If the selector applies only to queues of types other than the type of the object, the call succeeds with completion code MQCC_WARNING and reason code MQRC_SELECTOR_NOT_FOR_TYPE.
- If the queue being inquired is a cluster queue, the selectors that are valid depend on how the queue was resolved; see "Usage notes" on page 2017 for further details.

You can specify selectors in any order. Attribute values that correspond to integer attribute selectors (MQIA_* selectors) are returned in *IntAttrs* in the same order in which these selectors occur in *Selectors*. Attribute values that correspond to character attribute selectors (MQCA_* selectors) are returned in *CharAttrs* in the same order in which those selectors occur. MQIA_* selectors can be interleaved with the MQCA_* selectors; only the relative order within each type is important.

Note:

1. The integer and character attribute selectors are allocated within two different ranges; the MQIA_* selectors reside within the range MQIA_FIRST through MQIA_LAST, and the MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.

   For each range, the constants MQIA_LAST_USED and MQCA_LAST_USEDdefine the highest value that the queue manager accepts.
2. If all the MQIA_* selectors occur first, the same element numbers can be used to address corresponding elements in the *Selectors* and *IntAttrs* arrays.
3. If the *SelectorCount* parameter is zero, *Selectors* is not referred to. In this case, the parameter address passed by programs written in C or S/390 assembler might be null.

The attributes that can be inquired are listed in the following tables. For the MQCA_* selectors, the constant that defines the length in bytes of the resulting string in *CharAttrs* is provided in parentheses.

The tables that follow list the selectors, by object, in alphabetical order, as follows:

- Table 214 on page 2007 MQINQ attribute selectors for queues
- Table 215 on page 2009 MQINQ attribute selectors for namelists

- Table 216 on page 2009 MQINQ attribute selectors for process definitions
- Table 217 on page 2009 MQINQ attribute selectors for the queue manager

All selectors are supported on all IBM WebSphere MQ platforms, except where indicated in the **Note** column as follows:

Not z/OS
Supported on all platforms **except** z/OS

z/OS Supported **only** on z/OS

Table 214. MQINQ attribute selectors for queues

Selector	Length of field	Description	Note
MQCA_ALTERATION_DATE	MQ_DATE_LENGTH	Date of most-recent alteration	
MQCA_ALTERATION_TIME	MQ_TIME_LENGTH	Time of most-recent alteration	
MQCA_BACKOUT_REQ_Q_NAME	MQ_Q_NAME_LENGTH	Excessive backout requeue name	
MQCA_BASE_Q_NAME	MQ_Q_NAME_LENGTH	Name of queue that alias resolves to	
MQCA_CF_STRUC_NAME	MQ_CF_STRUC_NAME_LENGTH	Coupling-facility structure name	z/OS
MQCA_CLUS_CHL_NAME	MQ_CHANNEL_NAME_LENGTH	Name of the cluster-sender channel that uses this queue as a transmission queue.	Not z/OS
MQCA_CLUSTER_NAME	MQ_CLUSTER_NAME_LENGTH	Cluster name	
MQCA_CLUSTER_NAMELIST	MQ_NAMELIST_NAME_LENGTH	Cluster namelist	
MQCA_CREATION_DATE	MQ_CREATION_DATE_LENGTH	Queue creation date	
MQCA_CREATION_TIME	MQ_CREATION_TIME_LENGTH	Queue creation time	
MQCA_INITIATION_Q_NAME	MQ_Q_NAME_LENGTH	Initiation queue name	
MQCA_PROCESS_NAME	MQ_PROCESS_NAME_LENGTH	Name of process definition	
MQCA_Q_DESC	MQ_Q_DESC_LENGTH	Queue description	
MQCA_Q_NAME	MQ_Q_NAME_LENGTH	Queue name	
MQCA_REMOTE_Q_MGR_NAME	MQ_Q_MGR_NAME_LENGTH	Name of remote queue manager	
MQCA_REMOTE_Q_NAME	MQ_Q_NAME_LENGTH	Name of remote queue as known on remote queue manager	
MQCA_STORAGE_CLASS	MQ_STORAGE_CLASS_LENGTH	Name of storage class	z/OS
MQCA_TRIGGER_DATA	MQ_TRIGGER_DATA_LENGTH	Trigger data	
MQCA_XMIT_Q_NAME	MQ_Q_NAME_LENGTH	Transmission queue name	
MQIA_ACCOUNTING_Q	MQLONG	Controls collection of accounting data for queue	Not z/OS
MQIA_BACKOUT_THRESHOLD	MQLONG	Backout threshold	
MQIA_CLWL_Q_PRIORITY	MQLONG	Priority of queue	
MQIA_CLWL_Q_RANK	MQLONG	Rank of queue	
MQIA_CLWL_USEQ	MQLONG	Use remote queues	
MQIA_CURRENT_Q_DEPTH	MQLONG	Number of messages on queue	
MQIA_DEF_BIND	MQLONG	Default binding	
MQIA_DEF_INPUT_OPEN_OPTION	MQLONG	Default open-for-input option	
MQIA_DEF_PERSISTENCE	MQLONG	Default message persistence	
MQIA_DEF_PRIORITY	MQLONG	Default message priority	
MQIA_DEFINITION_TYPE	MQLONG	Queue definition type	

Table 214. MQINQ attribute selectors for queues (continued)

Selector	Length of field	Description	Note
MQIA_DIST_LISTS	MQLONG	Distribution list support	Not z/OS
MQIA_HARDEN_GET_BACKOUT	MQLONG	Whether to harden backout count	
MQIA_INDEX_TYPE	MQLONG	Type of index maintained for queue	z/OS
MQIA_INHIBIT_GET	MQLONG	Whether get operations are allowed	
MQIA_INHIBIT_PUT	MQLONG	Whether put operations are allowed	
MQIA_MAX_MSG_LENGTH	MQLONG	Maximum message length	
MQIA_MAX_Q_DEPTH	MQLONG	Maximum number of messages allowed on queue	
MQIA_MSG_DELIVERY_SEQUENCE	MQLONG	Whether message priority is relevant	
MQIA_NPM_CLASS	MQLONG	Level of reliability for nonpersistent messages	
MQIA_OPEN_INPUT_COUNT	MQLONG	Number of MQOPEN calls that have the queue open for input	
MQIA_OPEN_OUTPUT_COUNT	MQLONG	Number of MQOPEN calls that have the queue open for output	
MQIA_PROPERTY_CONTROL	MQLONG	Property control attribute	
MQIA_Q_DEPTH_HIGH_EVENT	MQLONG	Control attribute for queue depth high events	Not z/OS
MQIA_Q_DEPTH_HIGH_LIMIT	MQLONG	High limit for queue depth	Not z/OS
MQIA_Q_DEPTH_LOW_EVENT	MQLONG	Control attribute for queue depth low events	Not z/OS
MQIA_Q_DEPTH_LOW_LIMIT	MQLONG	Low limit for queue depth	Not z/OS
MQIA_Q_DEPTH_MAX_EVENT	MQLONG	Control attribute for queue depth max events	Not z/OS
MQIA_Q_SERVICE_INTERVAL	MQLONG	Limit for queue service interval	Not z/OS
MQIA_Q_SERVICE_INTERVAL_EVENT	MQLONG	Control attribute for queue service interval events	Not z/OS
MQIA_Q_TYPE	MQLONG	Queue type	
MQIA_QSG_DISP	MQLONG	Queue-sharing group disposition	z/OS
MQIA_RETENTION_INTERVAL	MQLONG	Queue retention interval	
MQIA_SCOPE	MQLONG	Queue definition scope	Not z/OS
MQIA_SHAREABILITY	MQLONG	Whether queue can be shared for input	
MQIA_STATISTICS_Q	MQLONG	Controls collection of statistics data for queue	Not z/OS
MQIA_TRIGGER_CONTROL	MQLONG	Trigger control	
MQIA_TRIGGER_DEPTH	MQLONG	Trigger depth	
MQIA_TRIGGER_MSG_PRIORITY	MQLONG	Threshold message priority for triggers	
MQIA_TRIGGER_TYPE	MQLONG	Trigger type	
MQIA_USAGE	MQLONG	Usage	

Table 215. MQINQ attribute selectors for namelists

Selector	Length of field	Description	Note
MQCA_ALTERATION_DATE	MQ_DATE_LENGTH	Date of most-recent alteration	
MQCA_ALTERATION_TIME	MQ_TIME_LENGTH	Time of most-recent alteration	
MQCA_NAMELIST_DESC	MQ_NAMELIST_DESC_LENGTH	Namelist description	
MQCA_NAMELIST_NAME	MQ_NAMELIST_NAME_LENGTH	Name of namelist object	
MQIA_NAMELIST_TYPE	MQLONG	Namelist type	z/OS
MQCA_NAMES	MQ_Q_NAME_LENGTH × Number of names in the list	Names in the namelist	
MQIA_NAME_COUNT	MQLONG	Number of names in the namelist	
MQIA_QSG_DISP	MQLONG	Queue-sharing group disposition	z/OS

Table 216. MQINQ attribute selectors for process definitions

Selector	Length of field	Description	Note
MQCA_ALTERATION_DATE	MQ_DATE_LENGTH	Date of most-recent alteration	
MQCA_ALTERATION_TIME	MQ_TIME_LENGTH	Time of most-recent alteration	
MQCA_APPL_ID	MQ_PROCESS_APPL_ID_LENGTH	Application identifier	
MQCA_ENV_DATA	MQ_PROCESS_ENV_DATA_LENGTH	Environment data	
MQCA_PROCESS_DESC	MQ_PROCESS_DESC_LENGTH	Description of process definition	
MQCA_PROCESS_NAME	MQ_PROCESS_NAME_LENGTH	Name of process definition	
MQCA_USER_DATA	MQ_PROCESS_USER_DATA_LENGTH	User data	
MQIA_APPL_TYPE	MQLONG	Application type	
MQIA_QSG_DISP	MQLONG	Queue-sharing group disposition	z/OS

Table 217. MQINQ attribute selectors for the queue manager

Selector	Length of field	Description	Note
MQCA_ALTERATION_DATE	MQ_DATE_LENGTH	Date of most-recent alteration	
MQCA_ALTERATION_TIME	MQ_TIME_LENGTH	Time of most-recent alteration	
MQCA_CHANNEL_AUTO_DEF_EXIT	MQ_EXIT_NAME_LENGTH	Automatic channel definition exit name	
MQCA_CHINIT_SERVICE_PARM		Reserved for use by IBM	
MQCA_CLUSTER_WORKLOAD_DATA	MQ_EXIT_DATA_LENGTH	Data passed to cluster workload exit	
MQCA_CLUSTER_WORKLOAD_EXIT	MQ_EXIT_NAME_LENGTH	Name of cluster workload exit	
MQCA_COMMAND_INPUT_Q_NAME	MQ_Q_NAME_LENGTH	System command input queue name	
MQCA_DEAD_LETTER_Q_NAME	MQ_Q_NAME_LENGTH	Name of dead-letter queue	
MQCA_DEF_XMIT_Q_NAME	MQ_Q_NAME_LENGTH	Default transmission queue name	
MQCA_DNS_GROUP	MQ_DNS_GROUP_NAME_LENGTH	Name of the group for the TCP listener that handles inbound transmissions for the queue-sharing group to join. The name applies when using Workload Manager Dynamic Domain Name Services.	z/OS
MQCA_IGQ_USER_ID	MQ_USER_ID_LENGTH	Intra-group queuing user identifier	z/OS

Table 217. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQCA_INSTALLATION_DESC	MQ_INSTALLATION_DESC_LENGTH	Description of the associated installation	Not z/OS. Not IBM i
MQCA_INSTALLATION_NAME	MQ_INSTALLATION_NAME_LENGTH	Name of the installation associated with the queue manager	Not z/OS. Not IBM i
MQCA_INSTALLATION_PATH	MQ_INSTALLATION_PATH_LENGTH	Path where the associatedIBM WebSphere MQ is installed	Not z/OS. Not IBM i
MQCA_LU_GROUP_NAME	MQ_LU_NAME_LENGTH	Generic LU name for the LU 6.2 listener that handles inbound transmissions for the queue-sharing group to use	z/OS
MQCA_LU_NAME	MQ_LU_NAME_LENGTH	Name of the LU to use for outbound LU 6.2 transmissions. Set this name to the same LU that the listener uses for inbound transmissions	z/OS
MQCA_LU62_ARM_SUFFIX	MQ_ARM_SUFFIX_LENGTH	Suffix of the SYS1.PARMLIB member APPCPM*xx*, that nominates the LUADD for this channel initiator	z/OS
MQCA_PARENT	MQ_Q_MGR_NAME_LENGTH	Name of a hierarchically connected queue manager that is nominated as the parent of this queue manager	
MQCA_Q_MGR_DESC	MQ_Q_MGR_DESC_LENGTH	Queue manager description	
MQCA_Q_MGR_IDENTIFIER	MQ_Q_MGR_IDENTIFIER_LENGT	Queue-manager identifier (H)	
MQCA_Q_MGR_NAME	MQ_Q_MGR_NAME_LENGTH	Name of local queue manager	
MQCA_QSG_NAME	MQ_QSG_NAME_LENGTH	Queue-sharing group name	z/OS
MQCA_REPOSITORY_NAME	MQ_CLUSTER_NAME_LENGTH	Name of cluster for which queue manager provides repository services	
MQCA_REPOSITORY_NAMELIST	MQ_NAMELIST_NAME_LENGTH	Name of namelist object containing names of clusters for which queue manager provides repository services	
MQCA_TCP_NAME	MQ_TCP_NAME_LENGTH	Name of the TCP/IP system that you are using	z/OS
MQIA_ACCOUNTING_CONN_OVERRIDE	MQLONG	Override accounting settings	Not z/OS
MQIA_ACCOUNTING_INTERVAL	MQLONG	How often to write intermediate accounting records	Not z/OS
MQIA_ACCOUNTING_MQI	MQLONG	Controls collection of accounting information for MQI data	Not z/OS
MQIA_ACCOUNTING_Q	MQLONG	Controls collection of accounting information for queues	Not z/OS
MQIA_ACTIVE_CHANNELS	MQLONG	Maximum number of channels that can be active at any time	z/OS

Table 217. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQIA_ADOPTNEWMCA_CHECK	MQLONG	Elements that are checked to determine whether to adopt an MCA. The check is performed when a new inbound channel is detected that has the same name as an MCA that is already active.	z/OS
MQIA_ADOPTNEWMCA_INTERVAL	MQLONG	Amount of time, in seconds, that the new channel waits for the orphaned channel to end	Not z/OS
MQIA_ADOPTNEWMCA_TYPE	MQLONG	Whether to restart an orphaned instance of an MCA of a particular channel type automatically when a new inbound channel request matching the AdoptNewMCACheck parameters is detected	z/OS
MQIA_AUTHORITY_EVENT	MQLONG	Control attribute for authority events	Not z/OS
MQIA_BRIDGE_EVENT	MQLONG	Control attribute for IMS bridge events	z/OS
MQIA_CHANNEL_AUTO_DEF	MQLONG	Control attribute for automatic channel definition	Not z/OS
MQIA_CHANNEL_AUTO_DEF_EVENT	MQLONG	Control attribute for automatic channel definition events	Not z/OS
MQIA_CHANNEL_EVENT	MQLONG	Control attribute for channel events	
MQIA_CHINIT_ADAPTERS	MQLONG	Number of adapter subtasks to use for processing IBM WebSphere MQ calls	z/OS
MQIA_CHINIT_DISPATCHERS	MQLONG	Number of dispatchers to use for the channel initiator	z/OS
MQIA_CHINIT_TRACE_AUTO_START	MQLONG	Whether to start channel initiator trace automatically	z/OS
MQIA_CHINIT_TRACE_TABLE_SIZE	MQLONG	Size of the trace data space (in MB) of the channel initiator	z/OS
MQIA_CLUSTER_WORKLOAD_LENGTH	MQLONG	Cluster workload length.	
MQIA_CLWL_MRU_CHANNELS	MQLONG	Number of most recently used channels for cluster workload balancing	
MQIA_CLWL_USEQ	MQLONG	Use remote queues	
MQIA_CODED_CHAR_SET_ID	MQLONG	Coded character set identifier	
MQIA_COMMAND_EVENT	MQLONG	Control attribute for command events	
MQIA_COMMAND_LEVEL	MQLONG	Command level supported by queue manager	
MQIA_CONFIGURATION_EVENT	MQLONG	Control attribute for configuration events	Not z/OS
MQIA_DEF_CLUSTER_XMIT_Q_TYPE	MQLONG	Default transmission queue type to be used for cluster-sender channels.	Not z/OS
MQIA_DIST_LISTS	MQLONG	Distribution list support	Not z/OS

Table 217. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQIA_DNS_WLM	MQLONG	Whether the TCP listener that handles inbound transmissions for the queue-sharing group registers with Workload Manager for Dynamic Domain Name Services	z/OS
MQIA_EXPIRY_INTERVAL	MQLONG	Interval between scans for expired messages	z/OS
MQIA_GROUP_UR	MQLONG	Control attribute for whether GROUP units of recovery are enabled for this queue manager. The GROUP unit of recovery disposition is only available if the queue manager is a member of a queue-sharing group	z/OS
MQIA_IGQ_PUT_AUTHORITY	MQLONG	Intra-group queuing put authority	z/OS
MQIA_INHIBIT_EVENT	MQLONG	Control attribute for inhibit events	Not z/OS
MQIA_INTRA_GROUP_QUEUING	MQLONG	Intra-group queuing support	z/OS
MQIA_LISTENER_TIMER	MQLONG	Time interval (in seconds) between IBM WebSphere MQ attempts to restart the listener if APPC or TCP/IP failed.	z/OS
MQIA_LOCAL_EVENT	MQLONG	Control attribute for local events	Not z/OS
MQIA_LOGGER_EVENT	MQLONG	Control attribute for inhibit events	Not z/OS
MQIA_LU62_CHANNELS	MQLONG	Maximum number of channels that can be current, or clients that can be connected, using the LU 6.2 transmission protocol	z/OS
MQIA_MSG_MARK_BROWSE_INTERVAL	MQLONG	Time interval (in milliseconds) after which the queue manager can automatically remove a mark from browse messages	
MQIA_MAX_CHANNELS	MQLONG	Maximum number of channels that can be current (including server-connection channels with connected clients)	z/OS
MQIA_MAX_HANDLES	MQLONG	Maximum number of handles	
MQIA_MAX_MSG_LENGTH	MQLONG	Maximum message length	
MQIA_MAX_PRIORITY	MQLONG	Maximum priority	
MQIA_MAX_UNCOMMITTED_MSGS	MQLONG	Maximum number of uncommitted messages within a unit of work	
MQIA_OUTBOUND_PORT_MAX	MQLONG	With MQIA_OUTBOUND_PORT_MIN, defines range of port numbers to use when binding outgoing channels	z/OS
MQIA_OUTBOUND_PORT_MIN	MQLONG	With MQIA_OUTBOUND_PORT_MAX, defines range of port numbers to use when binding outgoing channels	z/OS
MQIA_PERFORMANCE_EVENT	MQLONG	Control attribute for performance events	Not z/OS

Table 217. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQIA_PLATFORM	MQLONG	Platform on which the queue manager resides	
MQIA_PROT_POLICY_CAPABILITY	MQLONG	Indicates whether security capabilities of WebSphere MQ Advanced Message Security are available for a queue manager.	
MQIA_PUBSUB_MAXMSG_RETRY_COUNT	MQLONG	The number of attempts to reprocess a failed command message under sync point	
MQIA_PUBSUB_MODE	MQLONG	Whether the publish/subscribe engine and the queued publish/subscribe interface are running. Applications to publish or subscribe using the application programming interface require the publish/subscribe engine. Queues that are monitored by the queued publish/subscribe interface require the queued publish/subscribe interface to be running.	
MQIA_PUBSUB_NP_MSG	MQLONG	Whether to discard (or keep) an undelivered input message	
MQIA_PUBSUB_NP_RESP	MQLONG	Controls the behavior of undelivered response messages	
MQIA_PUBSUB_SYNC_PT	MQLONG	Whether only persistent (or all) messages are processed under sync point	
MQIA_QMGR_CFCONLOS	MQLONG	Specifies the action to be taken when the queue manager loses connectivity to the administration structure or any CF structures with CFCONLOS set to ASQMGR	z/OS
MQIA_RECEIVE_TIMEOUT	MQLONG	Approximately how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. The value is numeric, qualified by MQIA_RECEIVE_TIMEOUT_TYPE.	z/OS
MQIA_RECEIVE_TIMEOUT_MIN	MQLONG	Minimum time that a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state	z/OS
MQIA_RECEIVE_TIMEOUT_TYPE	MQLONG	Approximately how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. MQIA_RECEIVE_TIMEOUT_TYPE is the qualifier applied to MQIA_RECEIVE_TIMEOUT.	z/OS
MQIA_REMOTE_EVENT	MQLONG	Control attribute for remote events	Not z/OS
MQIA_SECURITY_CASE	MQLONG	Case of security profiles	z/OS
MQIA_SSL_EVENT	MQLONG	Control attribute for channel events	

Table 217. MQINQ attribute selectors for the queue manager (continued)

Selector	Length of field	Description	Note
MQIA_SSL_FIPS_REQUIRED	MQLONG	Use only FIPS-certified algorithms for cryptography	
MQIA_SSL_RESET_COUNT	MQLONG	SSL key reset count	
MQIA_START_STOP_EVENT	MQLONG	Control attribute for start stop events	Not z/OS
MQIA_STATISTICS_AUTO_CLUSSDR	MQLONG	Controls collection of statistics monitoring information for cluster sender channels	Not z/OS
MQIA_STATISTICS_CHANNEL	MQLONG	Controls collection of statistics data for channels	Not z/OS
MQIA_STATISTICS_INTERVAL	MQLONG	How often to write statistics monitoring data	Not z/OS
MQIA_STATISTICS_MQI	MQLONG	Controls collection of statistics monitoring information for queue manager	Not z/OS
MQIA_STATISTICS_Q	MQLONG	Controls collection of statistics data for queues	Not z/OS
MQIA_SYNCPOINT	MQLONG	sync point availability	
MQIA_TCP_CHANNELS	MQLONG	Maximum number of channels that can be current, or clients that can be connected, using the TCP/IP transmission protocol	z/OS
MQIA_TCP_KEEP_ALIVE	MQLONG	Whether to use the TCP KEEPALIVE facility to check that the other end of the connection is still available	z/OS
MQIA_TCP_STACK_TYPE	MQLONG	Whether the channel initiator can use only the TCP/IP address space specified in TCPNAME, or can optionally bind to any selected TCP/IP address	z/OS
MQIA_TRACE_ROUTE_RECORDING	MQLONG	Controls recording of trace-route information	z/OS
MQIA_TREE_LIFE_TIME	MQLONG	Lifetime of unused non-administrative topics	
MQIA_TRIGGER_INTERVAL	MQLONG	Trigger interval	

IntAttrCount

Type: MQLONG - input

This is the number of elements in the *IntAttrs* array. Zero is a valid value.

If IntAttrCount is at least the number of MQIA_* selectors in the *Selectors* parameter, all integer attributes requested are returned.

IntAttrs

Type: MQLONG ×*IntAttrCount* - output

This is an array of *IntAttrCount* integer attribute values.

Integer attribute values are returned in the same order as the MQIA_* selectors in the *Selectors* parameter. If the array contains more elements than the number of MQIA_* selectors, the excess elements are unchanged.

If *Hobj* represents a queue, but an attribute selector does not apply to that type of queue, the specific value MQIAV_NOT_APPLICABLE is returned. It is returned for the corresponding element in the *IntAttrs* array.

If the *IntAttrCount* or *SelectorCount* parameter is zero, *IntAttrs* is not referred to. In this case, the parameter address passed by programs written in C or S/390 assembler might be null.

CharAttrLength

Type: MQLONG - input

This is the length in bytes of the *CharAttrs* parameter.

CharAttrLength must be at least the sum of the lengths of the requested character attributes (see *Selectors*). Zero is a valid value.

CharAttrs

Type: MQCHAR × *CharAttrLength* - output

This is the buffer in which the character attributes are returned, concatenated together. The length of the buffer is given by the *CharAttrLength* parameter.

Character attributes are returned in the same order as the MQCA_* selectors in the *Selectors* parameter. The length of each attribute string is fixed for each attribute (see *Selectors*), and the value in it is padded to the right with blanks if necessary. You can provider a buffer larger than needed to contain all the requested character attributes and padding. The bytes beyond the last attribute value returned are unchanged.

If *Hobj* represents a queue, but an attribute selector does not apply to that type of queue, a character string consisting entirely of asterisks (*) is returned. The asterisk is returned as the value of that attribute in *CharAttrs*.

If the *CharAttrLength* or *SelectorCount* parameter is zero, *CharAttrs* is not referred to. In this case, the parameter address passed by programs written in C or S/390 assembler might be null.

CompCode

Type: MQLONG - output

The completion code:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CHAR_ATTRS_TOO_SHORT
(2008, X'7D8') Not enough space allowed for character attributes.

MQRC_INT_ATTR_COUNT_TOO_SMALL
(2022, X'7E6') Not enough space allowed for integer attributes.

MQRC_SELECTOR_NOT_FOR_TYPE
(2068, X'814') Selector not applicable to queue type.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CHAR_ATTR_LENGTH_ERROR
(2006, X'7D6') Length of character attributes not valid.

MQRC_CHAR_ATTRS_ERROR
(2007, X'7D7') Character attributes string not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INT_ATTR_COUNT_ERROR
(2021, X'7E5') Count of integer attributes not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'7E7') Integer attributes array not valid.

MQRC_NOT_OPEN_FOR_INQUIRE
(2038, X'7F6') Queue not open for inquire.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SELECTOR_COUNT_ERROR
(2065, X'811') Count of selectors not valid.

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED
(2066, X'812') Count of selectors too large.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes; see Reason codes

Usage notes

1. The values returned are a snapshot of the selected attributes. There is no guarantee that the attributes remain the same before the application can act upon the returned values.
2. When you open a model queue, a dynamic local queue is created. A dynamic local queue is created even if you open the model queue to inquire about its attributes.

   The attributes of the dynamic queue are largely the same as the attributes of the model queue at the time that the dynamic queue is created. If you then use the MQINQ call on this queue, the queue manager returns the attributes of the dynamic queue, and not the attributes of the model queue. See Table 220 on page 2137 for details of which attributes of the model queue are inherited by the dynamic queue.
3. If the object being inquired is an alias queue, the attribute values returned by the MQINQ call are the attributes of the alias queue. The are not the attributes of the base queue or topic to which the alias resolves.
4. If the object being inquired is a cluster queue, the attributes that can be inquired depend on how the queue is opened:
   - You can open a cluster queue for inquire plus one or more of the input, browse, or set operations. To do so, there must be a local instance of the cluster queue for the open to succeed. In this case, the attributes that can be inquired are the attributes that are valid for local queues.
   - If the cluster queue is opened for inquire alone, or inquire and output, only the attributes listed can be inquired. The *QType* attribute has the value MQQT_CLUSTER in this case:
     - MQCA_Q_DESC
     - MQCA_Q_NAME
     - MQIA_DEF_BIND
     - MQIA_DEF_PERSISTENCE

– MQIA_DEF_PRIORITY
– MQIA_INHIBIT_PUT
– MQIA_Q_TYPE

You can open the cluster queue with no fixed binding. You can open it with MQOO_BIND_NOT_FIXED specified on the MQOPEN call. Alternatively, specify MQOO_BIND_AS_Q_DEF, and set the *DefBind* attribute of the queue to MQBND_BIND_NOT_FIXED. If you open a cluster queue with no fixed binding, successive MQINQ calls for the queue might inquire different instances of the cluster queue. However, it is typical for all the instances have the same attribute values.

- An alias queue object can be defined for a cluster. Because TARGTYPE and TARGET are not cluster attributes, the process performing an MQOPEN process on the alias queue is not aware of the object to which the alias resolves.

  During the initial MQOPEN, the alias queue resolves to a queue manager and a queue in the cluster. Name resolution takes place again at the remote queue manager and it is here that the TARGTPYE of the alias queue is resolved.

  If the alias queue resolves to a topic alias, then publication of messages put to the alias queue takes place at this remote queue manager.

See Cluster queues

5. You might want to inquire a number of attributes, and then set some of them using the MQSET call. To program inquire and set efficiently, position the attributes to be set at the beginning of the selector arrays. If you do so, the same arrays with reduced counts can be used for MQSET.
6. If more than one of the warning situations arise (see the *CompCode* parameter), the reason code returned is the first one in the following list that applies:
   a. MQRC_SELECTOR_NOT_FOR_TYPE
   b. MQRC_INT_ATTR_COUNT_TOO_SMALL
   c. MQRC_CHAR_ATTRS_TOO_SHORT
7. The following topic have information about object attributes:
   - "Attributes for queues" on page 2135
   - "Attributes for namelists" on page 2171
   - "Attributes for process definitions" on page 2174
   - "Attributes for the queue manager" on page 2096

C invocation

```
MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
MQLONG   Selectors[n];    /* Array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
MQLONG   IntAttrs[n];     /* Array of integer attributes */
MQLONG   CharAttrLength;  /* Length of character attributes buffer */
MQCHAR   CharAttrs[n];    /* Character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQINQ' USING HCONN, HOBJ, SELECTORCOUNT, SELECTORS-TABLE,
                        INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,
                        CHARATTRS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN            PIC S9(9) BINARY.
**   Object handle
 01  HOBJ             PIC S9(9) BINARY.
**   Count of selectors
 01  SELECTORCOUNT    PIC S9(9) BINARY.
**   Array of attribute selectors
 01  SELECTORS-TABLE.
  02  SELECTORS        PIC S9(9) BINARY OCCURS n TIMES.
**   Count of integer attributes
 01  INTATTRCOUNT     PIC S9(9) BINARY.
**   Array of integer attributes
 01  INTATTRS-TABLE.
  02  INTATTRS         PIC S9(9) BINARY OCCURS n TIMES.
**   Length of character attributes buffer
 01  CHARATTRLENGTH   PIC S9(9) BINARY.
**   Character attributes
 01  CHARATTRS        PIC X(n).
**   Completion code
 01  COMPCODE         PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON           PIC S9(9) BINARY.
```

PL/I invocation

```
call MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
           IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn           fixed bin(31);  /* Connection handle */
dcl Hobj            fixed bin(31);  /* Object handle */
dcl SelectorCount   fixed bin(31);  /* Count of selectors */
dcl Selectors(n)    fixed bin(31);  /* Array of attribute selectors */
dcl IntAttrCount    fixed bin(31);  /* Count of integer attributes */
dcl IntAttrs(n)     fixed bin(31);  /* Array of integer attributes */
dcl CharAttrLength  fixed bin(31);  /* Length of character attributes
                                       buffer */
dcl CharAttrs       char(n);        /* Character attributes */
dcl CompCode        fixed bin(31);  /* Completion code */
dcl Reason          fixed bin(31);  /* Reason code qualifying
                                       CompCode */
```

High Level Assembler invocation

```
        CALL MQINQ,(HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT,  X
              INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN           DS  F      Connection handle
HOBJ            DS  F      Object handle
SELECTORCOUNT   DS  F      Count of selectors
SELECTORS       DS  (n)F   Array of attribute selectors
INTATTRCOUNT    DS  F      Count of integer attributes
INTATTRS        DS  (n)F   Array of integer attributes
CHARATTRLENGTH  DS  F      Length of character attributes buffer
CHARATTRS       DS  CL(n)  Character attributes
COMPCODE        DS  F      Completion code
REASON          DS  F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQINQ Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long   'Connection handle'
Dim Hobj           As Long   'Object handle'
Dim SelectorCount  As Long   'Count of selectors'
Dim Selectors      As Long   'Array of attribute selectors'
Dim IntAttrCount   As Long   'Count of integer attributes'
Dim IntAttrs       As Long   'Array of integer attributes'
Dim CharAttrLength As Long   'Length of character attributes buffer'
Dim CharAttrs      As String 'Character attributes'
Dim CompCode       As Long   'Completion code'
Dim Reason         As Long   'Reason code qualifying CompCode'
```

MQINQMP - Inquire message property:

The MQINQMP call returns the value of a property of a message.

Syntax

MQINQMP (*Hconn*, *Hmsg*, *InqPropOpts*, *Name*, *PropDesc*, *Type*, *ValueLength*, *Value*, *DataLength*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN then a valid connection must be established on the thread inquiring a property of the message handle otherwise the call fails with MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMSG - input

This is the message handle to be inquired. The value was returned by a previous **MQCRTMH** call.

InqPropOpts
Type: MQIMPO - input/output

See the MQIMPO data type for details.

Name
Type: MQCHARV - input/output

The name of the property to inquire.

If no property with this name can be found, the call fails with reason MQRC_PROPERTY_NOT_AVAILABLE.

You can use the wildcard character percent sign (%) at the end of the property name. The wildcard matches zero or more characters, including the period (.) character. This allows an application to inquire the value of many properties. Call MQINQMP with option MQIMPO_INQ_FIRST to get the first matching property and again with the option MQIMPO_INQ_NEXT to get the next matching property. When no more matching properties are available, the call fails with MQRC_PROPERTY_NOT_AVAILABLE. If the *ReturnedName* field of the InqPropOpts structure is initialized with an address or offset for the returned name of the property, this is completed on return from MQINQMP with the name of the property that has been matched. If the *VSBufSize* field of the *ReturnedName* in the InqPropOpts structure is less than the length of the returned property name the completion code is set MQCC_FAILED with reason MQRC_PROPERTY_NAME_TOO_BIG.

Properties that have known synonyms are returned as follows:

1. Properties with the prefix "mqps." are returned as the WebSphere MQ property name. For example, "MQTopicString" is the returned name rather than "mqps.Top"
2. Properties with the prefix "jms." or "mcd." are returned as the JMS header field name, for example, "JMSExpiration" is the returned name rather than "jms.Exp".
3. Properties with the prefix "usr." are returned without that prefix, for example, "Color" is returned rather than "usr.Color".

Properties with synonyms are only returned once.

In the C programming language, the following macro variables are defined for inquiring on all properties and then all properties that begin "usr.":

MQPROP_INQUIRE_ALL
Inquire on all properties of the message.

MQPROP_INQUIRE_ALL can be used in the following way:

```
MQCHARV Name = {MQPROP_INQUIRE_ALL};
```

MQPROP_INQUIRE_ALL_USR
Inquire on all properties of the message that start "usr.". The returned name is returned without the "usr." prefix.

If MQIMP_INQ_NEXT is specified but Name has changed since the previous call or this is the first call, then MQIMPO_INQ_FIRST is implied.

See Property names and Property name restrictions for further information about the use of property names.

PropDesc
Type: MQPD - output

This structure is used to define the attributes of a property, including what happens if the property is not supported, what message context the property belongs to, and what messages the property should be copied into. See MQPD for details of this structure.

Type
Type: MQLONG - input/output

On return from the MQINQMP call this parameter is set to the data type of *Value*. The data type can be any of the following:

MQTYPE_BOOLEAN
A boolean.

MQTYPE_BYTE_STRING
a byte string.

MQTYPE_INT8
An 8-bit signed integer.

MQTYPE_INT16
A 16-bit signed integer.

MQTYPE_INT32
A 32-bit signed integer.

MQTYPE_INT64
A 64-bit signed integer.

MQTYPE_FLOAT32
A 32-bit floating-point number.

MQTYPE_FLOAT64
A 64-bit floating-point number.

MQTYPE_STRING
A character string.

MQTYPE_NULL
The property exists but has a null value.

If the data type of the property value is not recognized then MQTYPE_STRING is returned and a string representation of the value is placed into the *Value* area. A string representation of the data type can be found in the *TypeString* field of the *InqPropOpts* parameter. A warning completion code is returned with reason MQRC_PROP_TYPE_NOT_SUPPORTED.

Additionally, if the option MQIMPO_CONVERT_TYPE is specified, conversion of the property value is requested. Use *Type* as an input to specify the data type that you want the property to be returned as. See the description of the MQIMPO_CONVERT_TYPE option of the MQIMPO structure for details of data type conversion.

If you do not request type conversion, you can use the following value on input:

MQTYPE_AS_SET
The value of the property is returned without converting its data type.

ValueLength
Type: MQLONG - input

The length in bytes of the Value area. Specify zero for properties that you do not require the value returned for. These could be properties which are designed by an application to have a null value or an empty string. Also specify zero if the MQIMPO_QUERY_LENGTH option has been specified; in this case no value is returned.

Value
Type: MQBYTEx*ValueLength* - output

This is the area to contain the inquired property value. The buffer should be aligned on a boundary appropriate for the value being returned. Failure to do so can result in an error when the value is later accessed.

If *ValueLength* is less than the length of the property value, as much of the property value as possible is moved into *Value* and the call fails with completion code MQCC_FAILED and reason MQRC_PROPERTY_VALUE_TOO_BIG.

The character set of the data in *Value* is given by the ReturnedCCSID field in the InqPropOpts parameter. The encoding of the data in *Value* is given by the ReturnedEncoding field in the InqPropOpts parameter.

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the *ValueLength* parameter is zero, *Value* is not referred to and its value passed by programs written in C or System/390 assembler can be null.

DataLength
Type: MQLONG - output

This is the length in bytes of the actual property value as returned in the *Value* area.

If *DataLength* is less than the property value length, *DataLength* is still filled in on return from the MQINQMP call. This allows the application to determine the size of the buffer required to accommodate the property value, and then reissue the call with a buffer of the appropriate size.

The following values can also be returned.

If the *Type* parameter is set to MQTYPE_STRING or MQTYPE_BYTE_STRING:

MQVL_EMPTY_STRING
The property exists but contains no characters or bytes.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_PROP_NAME_NOT_CONVERTED
(2492, X'09BC') Returned property name not converted.

MQRC_PROP_VALUE_NOT_CONVERTED
(2466, X'09A2') Property value not converted.

MQRC_PROP_TYPE_NOT_SUPPORTED
(2467, X'09A3') Property data type is not supported.

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'0852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'086D') Primary and home ASIDs differ.

MQRC_BUFFER_ERROR
(2004, X'07D4') Value parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Value length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DATA_LENGTH_ERROR
(2010, X'07DA') Data length parameter not valid.

MQRC_IMPO_ERROR
(2464, X'09A0') Inquire message property options structure not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07F8') Options not valid or not consistent.

MQRC_PD_ERROR
(2482, X'09B2') Property descriptor structure not valid.

MQRC_PROP_CONV_NOT_SUPPORTED
(2470, X'09A6') Conversion from the actual to requested data type not supported.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_PROPERTY_NAME_TOO_BIG
(2465, X'09A1') Property name too large for returned name buffer.

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7) Property not available.

MQRC_PROPERTY_VALUE_TOO_BIG
(2469, X'09A5') Property value too large for the Value area.

MQRC_PROP_NUMBER_FORMAT_ERROR
(2472, X'09A8') Number format error encountered in value data.

MQRC_PROPERTY_TYPE_ERROR
(2473, X'09A9') Invalid requested property type.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'0871') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'0893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQINQMP (Hconn, Hmsg, &InqPropOpts, &Name, &PropDesc, &Type,
ValueLength, Value, &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;       /* Connection handle */
MQHMSG  Hmsg;        /* Message handle */
MQIMPO InqPropOpts; /* Options that control the action of MQINQMP */
MQCHARV Name;        /* Property name */
MQPD    PropDesc;    /* Property descriptor */
MQLONG  Type;        /* Property data type */
MQLONG  ValueLength; /* Length in bytes of the Value area */
MQBYTE  Value[n];    /* Area to contain the property value */
MQLONG  DataLength;  /* Length of the property value */
MQLONG  CompCode;    /* Completion code */
MQLONG  Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQINQMP' USING HCONN, HMSG, INQMSGOPTS, NAME, PROPDESC, TYPE,
VALUELENGTH, VALUE, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN       PIC S9(9) BINARY.
**   Message handle
 01  HMSG      PIC S9(18) BINARY.
**   Options that control the action of MQINQMP
 01  INQMSGOPTS.
     COPY CMQIMPOV.
**   Property name
 01  NAME.
     COPY CMQCHRVV.
**   Property descriptor
 01  PROPDESC.
     COPY CMQPDV.
**   Property data type
 01  TYPE        PIC S9(9) BINARY.
**   Length in bytes of the VALUE area
 01  VALUELENGTH PIC S9(9) BINARY.
**   Area to contain the property value
 01  VALUE       PIC X(n).
**   Length of the property value
 01  DATALENGTH  PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE    PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON      PIC S9(9) BINARY.
```

PL/I invocation

```
call MQINQMP (Hconn, Hmsg, InqPropOpts, Name, PropDesc, Type,
ValueLength, Value, DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31);  /* Connection handle */
dcl Hmsg        fixed bin(63);  /* Message handle */
dcl InqPropOpts like MQIMPO;    /* Options that control the action of MQINQMP */
dcl Name        like MQCHARV;   /* Property name */
dcl PropDesc    like MQPD;      /* Property descriptor */
dcl Type        fixed bin (31); /* Property data type */
dcl ValueLength fixed bin (31); /* Length in bytes of the Value area */
dcl Value       char (n);       /* Area to contain the property value */
dcl DataLength  fixed bin (31); /* Length of the property value */
dcl CompCode    fixed bin (31); /* Completion code */
dcl Reason      fixed bin (31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQINQMP,(HCONN,HMSG,INQMSGOPTS,NAME,PROPDESC,TYPE,
VALUELENGTH,VALUE,DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS         F     Connection handle
HMSG         DS         D     Message handle
INQMSGOPTS   CMQIMPOA   ,     Options that control the action of MQINQMP
NAME         CMQCHRVA   ,     Property name
PROPDESC     CMQPDA     ,     Property descriptor
TYPE         DS         F     Property data type
VALUELENGTH  DS         F     Length in bytes of the VALUE area
VALUE        DS         CL(n) Area to contain the property value
DATALENGTH   DS         F     Length of the property value
COMPCODE     DS         F     Completion code
REASON       DS         F     Reason code qualifying COMPCODE
```

MQMHBUF - Convert message handle into buffer:

The MQMHBUF call converts a message handle into a buffer and is the inverse of the MQBUFMH call.

Syntax

MQMHBUF (*Hconn*, *Hmsg*, *MsgHBufOpts*, *Name*, *MsgDesc*, *BufferLength*, *Buffer*, *DataLength*, *CompCode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter.

If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread deleting the message handle. If a valid connection is not established, the call fails with MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMSG - input

This is the message handle for which a buffer is required. The value was returned by a previous MQCRTMH call.

MsgHBufOpts
Type: MQMHBO - input

The MQMHBO structure allows applications to specify options that control how buffers are produced from message handles.

See “MQMHBO - Message handle to buffer options” on page 1765 for details.

Name
Type: MQCHARV - input

The name of the property or properties to put into the buffer.

If no property matching the name can be found, the call fails with MQRC_PROPERTY_NOT_AVAILABLE.

You can use a wildcard to put more than one property into the buffer. To do this, use the wildcard character '%' at the end of the property name. This wildcard matches zero or more characters, including the '.' character.

In the C programming language, the following macro variables are defined for inquiring on all properties and all properties that begin 'usr':

MQPROP_INQUIRE_ALL
Put all properties of the message into the buffer

MQPROP_INQUIRE_ALL_USR
Put all properties of the message that start with the characters 'usr.' into the buffer.

See Property names and Property name restrictions for further information about the use of property names.

MsgDesc
Type: MQMD - input/output

The *MsgDesc* structure describes the contents of the buffer area.

On output, the *Encoding*, *CodedCharSetId* and *Format* fields are set to correctly describe the encoding, character set identifier, and format of the data in the buffer area as written by the call.

Data in this structure is in the character set and encoding of the application.

BufferLength
Type: MQLONG - input

BufferLength is the length of the Buffer area, in bytes.

Buffer
Type: MQBYTExBufferLength - output

Buffer defines the area to contain the message properties. You must align the buffer on a 4-byte boundary.

If *BufferLength* is less than the length required to store the properties in *Buffer*, MQMHBUF fails with MQRC_PROPERTY_VALUE_TOO_BIG.

The contents of the buffer can change even if the call fails.

DataLength
Type: MQLONG - output

DataLength is the length, in bytes, of the returned properties in the buffer. If the value is zero, no properties matched the value given in *Name* and the call fails with reason code MQRC_PROPERTY_NOT_AVAILABLE.

If *BufferLength* is less than the length required to store the properties in the buffer, the MQMHBUF call fails with MQRC_PROPERTY_VALUE_TOO_BIG, but a value is still entered into *DataLength*. This allows the application to determine the size of the buffer required to accommodate the properties, and then reissue the call with the required *BufferLength*.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_MHBO_ERROR
(2501, X'095C') Message handle to buffer options structure not valid.

MQRC_BUFFER_ERROR
(2004, X'07D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_CONNECTION_BROKEN
(2009, X'07D9') Connection to queue manager lost.

MQRC_DATA_LENGTH_ERROR
(2010, X'07DA') Data length parameter not valid.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle not valid.

MQRC_MD_ERROR
(2026, X'07EA') Message descriptor not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Property name is not valid.

MQRC_PROPERTY_NOT_AVAILABLE
(2471, X'09A7') Property not available.

MQRC_PROPERTY_VALUE_TOO_BIG
(2469, X'09A5') BufferLength value is too small to contain specified properties.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQMHBUF (Hconn, Hmsg, &MsgHBufOpts, &Name, &MsgDesc, BufferLength, Buffer,
         &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;        /* Connection handle */
MQHMSG  Hmsg;         /* Message handle */
MQMHBO  MsgHBufOpts;  /* Options that control the action of MQMHBUF */
MQCHARV Name;         /* Property name */
MQMD    MsgDesc;      /* Message descriptor */
MQLONG  BufferLength; /* Length in bytes of the Buffer area */
MQBYTE  Buffer[n];    /* Area to contain the properties */
MQLONG  DataLength;   /* Length of the properties */
MQLONG  CompCode;     /* Completion code */
MQLONG  Reason;       /* Reason code qualifying CompCode */
```

Usage notes

MQMHBUF converts a message handle into a buffer.

You can use it with an MQGET API exit to access certain properties, using the message property APIs, and then pass these in a buffer back to an application designed to use MQRFH2 headers rather than message handles.

This call is the inverse of the MQBUFMH call, which you can use to parse message properties from a buffer into a message handle.

COBOL invocation

```
     CALL 'MQMHBUF' USING HCONN, HMSG, MSGHBUFOPTS, NAME, MSGDESC,
                          BUFFERLENGTH, BUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Message handle
 01  HMSG          PIC S9(18) BINARY.
**   Options that control the action of MQMHBUF
 01  MSGHBUFOPTS.
     COPY CMQMHBOV.
**   Property name
 01  NAME
     COPY CMQCHRVV.
**   Message descriptor
 01  MSGDESC
     COPY CMQMDV.
**   Length in bytes of the Buffer area */
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Area to contain the properties
 01  BUFFER        PIC X(n).
**   Length of the properties
 01  DATALENGTH    PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQMHBUF (Hconn, Hmsg, MsgHBufOpts, Name, MsgDesc, BufferLength, Buffer,
              DataLength, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn        fixed bin(31); /* Connection handle */
dcl Hmsg         fixed bin(63); /* Message handle */
dcl MsgHBufOpts  like MQMHBO;   /* Options that control the action of MQMHBUF */
dcl Name         like MQCHARV;  /* Property name */
dcl MsgDesc      like MQMD;     /* Message descriptor */
dcl BufferLength fixed bin(31); /* Length in bytes of the Buffer area */
dcl Buffer       char(n);       /* Area to contain the properties */
dcl DataLength   fixed bin(31); /* Length of the properties */
dcl CompCode     fixed bin(31); /* Completion code */
dcl Reason       fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQMHBUF,(HCONN,HMSG,MSGHBUFOPTS,NAME,MSGDESC,BUFFERLENGTH,
                BUFFER,DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS       F      Connection handle
HMSG         DS       D      Message handle
MSGHBUFOPTS  CMQMHBOA ,      Options that control the action of MQMHBUF
NAME         CMQCHRVA ,      Property name
MSGDESC      CMQMDA   ,      Message descriptor
BUFFERLENGTH DS       F      Length in bytes of the BUFFER area
BUFFER       DS       CL(n)  Area to contain the properties
```

```
DATALENGTH   DS       F      Length of the properties
COMPCODE     DS       F      Completion code
REASON       DS       F      Reason code qualifying COMPCODE
```

MQOPEN - Open object:

The MQOPEN call establishes access to an object.

The following types of object are valid:

- Queue (including distribution lists)
- Namelist
- Process definition
- Queue manager
- Topic

Syntax

MQOPEN (*Hconn*, *ObjDesc*, *Options*, *Hobj*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

ObjDesc

Type: MQOD - input/output

This is a structure that identifies the object to be opened; see "MQOD - Object descriptor" on page 1768 for details.

If the *ObjectName* field in the *ObjDesc* parameter is the name of a model queue, a dynamic local queue is created with the attributes of the model queue; this happens whatever options you specify on the *Options* parameter. Subsequent operations using the *Hobj* returned by the MQOPEN call are performed on the new dynamic queue, and not on the model queue. This is true even for the MQINQ and MQSET calls. The name of the model queue in the *ObjDesc* parameter is replaced with the name of the dynamic queue created. The type of the dynamic queue is determined by the value of the *DefinitionType* attribute of the model queue (see "Attributes for queues" on page 2135). For information about the close options applicable to dynamic queues, see the description of the MQCLOSE call.

Options

Type: MQLONG - input

You must specify at least one of the following options:

- MQOO_BROWSE
- MQOO_INPUT_* (only one of these)
- MQOO_INQUIRE
- MQOO_OUTPUT
- MQOO_SET

- MQOO_BIND_* (only one of these)

See the following table for details of these options; other options can be specified as required. If more than one option is required, the values can be:

- Added together (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations).

Combinations that are not valid are noted; all other combinations are valid. Only options that are applicable to the type of object specified by *ObjDesc* are allowed. The following table shows valid MQOPEN options for queries and topics.

Option	Alias [1]	Local and Model	Remote	Nonlocal Cluster	Distribution list	Topic
MQOO_INPUT_AS_Q_DEF	Yes	Yes	No	No	No	No
MQOO_INPUT_SHARED	Yes	Yes	No	No	No	No
MQOO_INPUT_EXCLUSIVE	Yes	Yes	No	No	No	No
MQOO_OUTPUT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_BROWSE	Yes	Yes	No	No	No	No
MQOO_CO_OP	Yes	Yes	No	No	No	No
MQOO_INQUIRE	Yes	Yes	[2]	Yes	No	No
MQOO_SET	Yes	Yes	[2]	No	No	No
MQOO_BIND_ON_OPEN [3]	Yes	Yes	Yes	Yes	Yes	No
MQOO_BIND_NOT_FIXED [3]	Yes	Yes	Yes	Yes	Yes	No
MQOO_BIND_ON_GROUP [3]	Yes	Yes	Yes	Yes	Yes	No
MQOO_BIND_AS_Q_DEF [3]	Yes	Yes	Yes	Yes	Yes	No
MQOO_SAVE_ALL_CONTEXT	Yes	Yes	No	No	No	No
MQOO_PASS_IDENTITY_CONTEXT	Yes	Yes	Yes	Yes	Yes	[4]
MQOO_PASS_ALL_CONTEXT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_SET_IDENTITY_CONTEXT	Yes	Yes	Yes	Yes	Yes	[4]
MQOO_SET_ALL_CONTEXT	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_NO_READ_AHEAD	Yes	Yes	No	No	No	No
MQOO_READ_AHEAD	Yes	Yes	No	No	No	No
MQOO_READ_AHEAD_AS_Q_DEF	Yes	Yes	No	No	No	No
MQOO_ALTERNATE_USER_AUTHORITY	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_FAIL_IF_QUIESCING	Yes	Yes	Yes	Yes	Yes	Yes
MQOO_RESOLVE_LOCAL_Q	Yes	Yes	Yes	Yes	No	No
MQOO_RESOLVE_LOCAL_TOPIC	No	No	No	No	No	Yes
MQOO_NO_MULTICAST	No	No	No	No	No	Yes

Note:

1. The validity of options for aliases depends on the validity of the option for the queue to which the alias resolves.
2. This option is valid only for the local definition of a remote queue.
3. This option can be specified for any queue type, but is ignored if the queue is not a cluster queue. However, the *DefBind* queue attribute overrides the base queue even when the alias queue is not in a cluster.
4. These attributes can be used with a topic, but affect only the context set for the retained message, not the context fields sent to any subscriber.

Access options: The following options control the type of operations that can be performed on the object:

MQOO_INPUT_AS_Q_DEF
Open queue to get messages using queue-defined default.

The queue is opened for use with subsequent MQGET calls. The type of access is either shared or exclusive, depending on the value of the *DefInputOpenOption* queue attribute; see "Attributes for queues" on page 2135 for details.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_OUTPUT

Open queue to put messages, or a topic or topic string to publish messages.

The queue or topic is opened for use with subsequent MQPUT calls.

An MQOPEN call with this option can succeed even if the *InhibitPut* queue attribute is set to MQQA_PUT_INHIBITED (although subsequent MQPUT calls fail while the attribute is set to this value).

This option is valid for all types of queue, including distribution lists, and topics.

The following notes apply to these options:

- Only one of these options can be specified.
- An MQOPEN call with one of these options can succeed even if the *InhibitGet* queue attribute is set to MQQA_GET_INHIBITED (although subsequent MQGET calls fail while the attribute is set to this value).
- If the queue is defined as not being shareable (that is, the *Shareability* queue attribute has the value MQQA_NOT_SHAREABLE), attempts to open the queue for shared access are treated as attempts to open the queue with exclusive access.
- If an alias queue is opened with one of these options, the test for exclusive use (or for whether another application has exclusive use) is against the base queue to which the alias resolves.
- These options are not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_BROWSE
Open queue to browse messages.

The queue is opened for use with subsequent MQGET calls with one of the following options:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT

- MQGMO_BROWSE_MSG_UNDER_CURSOR

This is allowed even if the queue is currently open for MQOO_INPUT_EXCLUSIVE. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, and positions it logically before the first message on the queue; see MQGMO - Options field for further information.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues. It is also not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_CO_OP
Open as a cooperating member of the set of handles.

This option is valid only with the MQOO_BROWSE option. If it is specified without MQOO_BROWSE, MQOPEN returns with MQRC_OPTIONS_ERROR.

The handle returned is considered to be a member of a cooperating set of handles for subsequent MQGET calls with one of the following options:

- MQGMO_MARK_BROWSE_CO_OP
- MQGMO_UNMARKED_BROWSE_MSG
- MQGMO_UNMARK_BROWSE_CO_OP

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_INQUIRE
Open object to inquire attributes.

The queue, namelist, process definition, or queue manager is opened for use with subsequent MQINQ calls.

This option is valid for all types of object other than distribution lists. It is not valid if *ObjectQMgrName* is the name of a queue manager alias; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_SET
Open queue to set attributes.

The queue is opened for use with subsequent MQSET calls.

This option is valid for all types of queue other than distribution lists. It is not valid if *ObjectQMgrName* is the name of a local definition of a remote queue; this is true even if the value of the *RemoteQMgrName* attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

Binding options: The following options apply when the object being opened is a cluster queue; these options control the binding of the queue handle to an instance of the cluster queue:

MQOO_BIND_ON_OPEN
The local queue manager binds the queue handle to an instance of the destination queue when the queue is opened. As a result, all messages put using this handle are sent to the same instance of the destination queue, and by the same route.

This option is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BIND_NOT_FIXED
This stops the local queue manager binding the queue handle to an instance of the destination queue. As a result, successive MQPUT calls using this handle send the messages

to *different* instances of the destination queue, or to the same instance but by different routes. It also allows the instance selected to be changed later by the local queue manager, by a remote queue manager, or by a message channel agent (MCA), according to network conditions.

Note: Client and server applications that need to exchange a *series* of messages to complete a transaction must not use MQOO_BIND_NOT_FIXED (or MQOO_BIND_AS_Q_DEF when *DefBind* has the value MQBND_BIND_NOT_FIXED), because successive messages in the series might be sent to different instances of the server application.

If MQOO_BROWSE or one of the MQOO_INPUT_* options is specified for a cluster queue, the queue manager is forced to select the local instance of the cluster queue. As a result, the binding of the queue handle is fixed, even if MQOO_BIND_NOT_FIXED is specified.

If MQOO_INQUIRE is specified with MQOO_BIND_NOT_FIXED, successive MQINQ calls using that handle might inquire different instances of the cluster queue, although typically all the instances have the same attribute values.

MQOO_BIND_NOT_FIXED is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

This option is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BIND_AS_Q_DEF
The local queue manager binds the queue handle in the way defined by the *DefBind* queue attribute. The value of this attribute is either MQBND_BIND_ON_OPEN, MQBND_BIND_NOT_FIXED, or MQBND_BIND_ON_GROUP.

MQOO_BIND_AS_Q_DEF is the default when MQOO_BIND_ON_OPEN, MQOO_BIND_NOT_FIXED, or MQOO_BIND_ON_GROUP is not specified.

MQOO_BIND_AS_Q_DEF aids program documentation. It is not intended that this option is used with either of the other two bind options, but because its value is zero such use cannot be detected.

Context options: The following options control the processing of message context:

MQOO_SAVE_ALL_CONTEXT
Context information is associated with this queue handle. This information is set from the context of any message retrieved using this handle. For more information about message context, see Message context and Controlling context information.

This context information can be passed to a message that is then put on a queue using the MQPUT or MQPUT1 calls. See the MQPMO_PASS_IDENTITY_CONTEXT and MQPMO_PASS_ALL_CONTEXT options described in “MQPMO - Put-message options” on page 1791.

Until a message has been successfully retrieved, context cannot be passed to a message being put on a queue.

A message retrieved using one of the MQGMO_BROWSE_* browse options does not have its context information saved (although the context fields in the *MsgDesc* parameter are set after a browse).

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues. One of the MQOO_INPUT_* options must be specified.

MQOO_PASS_IDENTITY_CONTEXT
This allows the MQPMO_PASS_IDENTITY_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. For more information about message context, see Message context and Controlling context information.

The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_PASS_ALL_CONTEXT
This allows the MQPMO_PASS_ALL_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity and origin context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. For more information about message context, see Message context and Controlling context information .

This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_SET_IDENTITY_CONTEXT
This allows the MQPMO_SET_IDENTITY_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity context information contained in the *MsgDesc* parameter specified on the MQPUT or MQPUT1 call. For more information about message context, see Message context and Controlling context information .

This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

MQOO_SET_ALL_CONTEXT
This allows the MQPMO_SET_ALL_CONTEXT option to be specified in the *PutMsgOpts* parameter when a message is put on a queue; this gives the message the identity and origin context information contained in the *MsgDesc* parameter specified on the MQPUT or MQPUT1 call. For more information about message context, see Message context and Controlling context information .

This option implies the following options, which need not therefore be specified:

- MQOO_PASS_IDENTITY_CONTEXT
- MQOO_PASS_ALL_CONTEXT
- MQOO_SET_IDENTITY_CONTEXT

The MQOO_OUTPUT option must be specified.

This option is valid for all types of queue, including distribution lists.

Read ahead options:

When you call MQOPEN with MQOO_READ_AHEAD, the WebSphere MQ client only enables readahead if certain conditions are met. These conditions include:

- Both the client and remote queue manager must be at WebSphere MQ Version 7 or later.
- The client application must be compiled and linked against the threaded WebSphere MQ MQI client libraries.
- The client channel must be using TCP/IP protocol
- The channel must have a non-zero SharingConversations (SHARECNV) setting in both the client and server channel definitions.

The following options control whether non-persistent messages are sent to the client before an application requests them. The following notes apply to the read ahead options:

- Only one of these options can be specified.
- These options are valid only for local, alias, and model queues. They are not valid for remote queues, distribution lists, topics or queue managers.
- These options are only applicable when one of MQOO_BROWSE, MQOO_INPUT_SHARED and MQOO_INPUT_EXCLUSIVE are also specified although it is not an error to specify these options with MQOO_INQUIRE or MQOO_SET.
- If the application is not running as a IBM WebSphere MQ client, these options are ignored.

MQOO_NO_READ_AHEAD
Non-persistent messages are not sent the client before an application requests them.

MQOO_READ_AHEAD
Non-persistent messages are sent to the client before an application requests them.

MQOO_READ_AHEAD_AS_Q_DEF
Read ahead behavior is determined by the default read ahead attribute of the queue being opened. This is the default value.

Other options: The following options control authorization checking, what happens when the queue manager is quiescing, whether to resolve the local queue name, and multicast:

MQOO_ALTERNATE_USER_AUTHORITY
The *AlternateUserId* field in the *ObjDesc* parameter contains a user identifier to use to validate this MQOPEN call. The call can succeed only if this *AlternateUserId* is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so. This does not apply to any context options specified, however, which are always checked against the user identifier under which the application is running.

This option is valid for all types of object.

MQOO_FAIL_IF_QUIESCING
The MQOPEN call fails if the queue manager is in quiescing state.

On z/OS, for a CICS or IMS application, this option also forces the MQOPEN call to fail if the connection is in quiescing state.

This option is valid for all types of object.

For information about client channels see Overview of IBM WebSphere MQ MQI clients .

MQOO_RESOLVE_LOCAL_Q
Fill the ResolvedQName in the MQOD structure with the name of the local queue that was opened. Similarly, the ResolvedQMgrName is filled with the name of the local queue manager hosting the local queue. If the MQOD structure is less than Version 3, MQOO_RESOLVE_LOCAL_Q is ignored with no error being returned.

The local queue is always returned when either a local, alias, or model queue is opened, but this is not the case when, for example, a remote queue or a non-local cluster queue is opened without the MQOO_RESOLVE_LOCAL_Q option; the ResolvedQName and ResolvedQMgrName are filled with the RemoteQName and RemoteQMgrName found in the remote queue definition, or similarly with the chosen remote cluster queue.

If you specify MQOO_RESOLVE_LOCAL_Q when opening, for example, a remote queue, ResolvedQName is the transmission queue to which messages are put. The ResolvedQMgrName is filled with the name of the local queue manager hosting the transmission queue.

If you are authorized for browse, input, or output on a queue, you have the required authority to specify this flag on the MQOPEN call. No special authority is needed.

This option is valid only for queues and queue managers.

MQOO_RESOLVE_LOCAL_TOPIC
Fill the ResolvedQName in the MQOD structure with the name of the administrative topic opened.

MQOO_NO_MULTICAST
Publication messages are not sent using multicast.

This option is valid only with the MQOO_OUTPUT option. If it is specified without MQOO_OUTPUT, MQOPEN returns with MQRC_OPTIONS_ERROR.

This option is valid only for a topic.

Hobj
Type: MQHOBJ - output

This handle represents the access that has been established to the object. It must be specified on subsequent IBM WebSphere MQ calls that operate on the object. It ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates.

The scope of the object handle returned is the same as the scope of the connection handle specified on the call. See MQCONN - Hconn parameter for information about handle scope.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') Coupling facility not available.

MQRC_CF_STRUC_AUTH_FAILED
(2348, X'92C') Coupling-facility structure authorization check failed.

MQRC_CF_STRUC_ERROR
(2349, X'92D') Coupling-facility structure not valid.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_PUT_INHIBITED
(2268, X'8DC') Put calls inhibited for all queues in cluster.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'8DD') Cluster resource error.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') Db2 subsystem not available.

MQRC_DEF_XMIT_Q_TYPE_ERROR
(2198, X'896') Default transmission queue not local.

MQRC_DEF_XMIT_Q_USAGE_ERROR
(2199, X'897') Default transmission queue usage error.

MQRC_DYNAMIC_Q_NAME_ERROR
(2011, X'7DB') Name of dynamic queue not valid.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'7E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_NAME_IN_USE
(2201, X'899') Name in use.

MQRC_NAME_NOT_VALID_FOR_TYPE
(2194, X'892') Object name not valid for object type.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OBJECT_ALREADY_EXISTS
(2100, X'834') Object exists.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OBJECT_IN_USE
(2042, X'7FA') Object already open with conflicting options.

MQRC_OBJECT_LEVEL_INCOMPATIBLE
(2360, X'938') Object level not compatible.

MQRC_OBJECT_NAME_ERROR
(2152, X'868') Object name not valid.

MQRC_OBJECT_NOT_UNIQUE
(2343, X'927') Object not unique.

MQRC_OBJECT_Q_MGR_NAME_ERROR
(2153, X'869') Object queue-manager name not valid.

MQRC_OBJECT_RECORDS_ERROR
(2155, X'86B') Object records not valid.

MQRC_OBJECT_STRING_ERROR
(2441, X'0989') Objectstring field not valid

MQRC_OBJECT_TYPE_ERROR
(2043, X'7FB') Object type not valid.

MQRC_OD_ERROR
(2044, X'7FC') Object descriptor structure not valid.

MQRC_OPTION_NOT_VALID_FOR_TYPE
(2045, X'7FD') Option not valid for object type.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_PAGESET_FULL
(2192, X'890') External storage medium is full.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_TYPE_ERROR
(2057, X'809') Queue type not valid.

MQRC_RECS_PRESENT_ERROR
(2154, X'86A') Number of records present not valid.

MQRC_REMOTE_Q_NAME_ERROR
(2184, X'888') Remote queue name not valid.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SELECTOR_SYNTAX_ERROR
2459 (X'099B') An MQOPEN, MQPUT1 or MQSUB call was issued but a selection string was specified which contained a syntax error.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.

MQRC_UNKNOWN_DEF_XMIT_Q
(2197, X'895') Unknown default transmission queue.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'826') Unknown object queue manager.

MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.

MQRC_UNKNOWN_XMIT_Q
(2196, X'894') Unknown transmission queue.

MQRC_WRONG_CF_LEVEL
(2366, X'93E') Coupling-facility structure is wrong level.

MQRC_XMIT_Q_TYPE_ERROR
(2091, X'82B') Transmission queue not local.

MQRC_XMIT_Q_USAGE_ERROR
(2092, X'82C') Transmission queue with wrong usage.

For detailed information about these codes, see:
- Reason codes for all other IBM WebSphere MQ platforms except z/OS.

General usage notes

1. The object opened is one of the following:
   - A queue to:
     - Get or browse messages (using the MQGET call)
     - Put messages (using the MQPUT call)
     - Inquire about the attributes of the queue (using the MQINQ call)
     - Set the attributes of the queue (using the MQSET call)

     If the queue named is a model queue, a dynamic local queue is created. See the *ObjDesc* parameter described in "MQOPEN - Open object" on page 2030.

     A distribution list is a special type of queue object that contains a list of queues. It can be opened to put messages, but not to get or browse messages, or to inquire or set attributes. See usage note 8 for further details.

     A queue that has `QSGDISP(GROUP)` is a special type of queue definition that cannot be used with the MQOPEN or MQPUT1 calls.
   - A namelist to inquire about the names of the queues in the list (using the MQINQ call).
   - A process definition to inquire about the process attributes (using the MQINQ call).
   - The queue manager to inquire about the attributes of the local queue manager (using the MQINQ call).
   - A topic to publish a message (using the MQPUT call)
2. An application can open the same object more than once. A different object handle is returned for each open. Each handle that is returned can be used for the functions for which the corresponding open was performed.
3. If the object being opened is a queue other than a cluster queue, all name resolution within the local queue manager takes place at the time of the MQOPEN call. This can include:
   - Resolution of the name of a local definition of a remote queue to the name of the remote queue manager, and the name by which the queue is known at the remote queue manager
   - Resolution of the remote queue-manager name to the name of a local transmission queue
   - (z/OS only) Resolution of the remote queue-manager name to the name of the shared transmission queue used by the IGQ agent (applies only if the local and remote queue managers belong to the same queue-sharing group)
   - Alias resolution to the name of a base queue or a topic object.

   However, be aware that subsequent MQINQ or MQSET calls for the handle relate solely to the name that has been opened, and not to the object resulting after name resolution has occurred. For example, if the object opened is an alias, the attributes returned by the MQINQ call are the attributes of the alias, not the attributes of the base queue or a topic object to which the alias resolves.

   If the object being opened is a cluster queue, name resolution can occur at the time of the MQOPEN call, or be deferred until later. The point at which resolution occurs is controlled by the MQOO_BIND_* options specified on the MQOPEN call:

- MQOO_BIND_ON_OPEN
- MQOO_BIND_NOT_FIXED
- MQOO_BIND_AS_Q_DEF
- MQOO_BIND_ON_GROUP

See Name resolution for more information about name resolution for cluster queues.

4. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, for use with MQGET calls that specify the object handle and one of the browse options. This allows the queue to be scanned without altering its contents. A message that has been found by browsing can be removed from the queue by using the MQGMO_MSG_UNDER_CURSOR option.

   Multiple browse cursors can be active for a single application by issuing several MQOPEN requests for the same queue.

5. Applications started by a trigger monitor are passed the name of the queue that is associated with the application when the application is started. This queue name can be specified in the *ObjDesc* parameter to open the queue. See "MQTMC2 - Trigger message 2 (character format)" on page 1904 for further details.

6. On IBM i, applications running in compatibility mode are connected automatically to the queue manager by the first MQOPEN call issued by the application (if the application has not already connected to the queue manager by using the MQCONN call).

   Applications not running in compatibility mode must issue the MQCONN or MQCONNX call to connect to the queue manager explicitly, before using the MQOPEN call to open an object.

Read ahead options

When you call MQOPEN with MQOO_READ_AHEAD, the WebSphere MQ client only enables readahead if certain conditions are met. These conditions include:

- Both the client and remote queue manager must be at WebSphere MQ Version 7 or later.
- The client application must be compiled and linked against the threaded WebSphere MQ MQI client libraries.
- The client channel must be using TCP/IP protocol
- The channel must have a non-zero SharingConversations (SHARECNV) setting in both the client and server channel definitions.

The following notes apply to the use of read ahead options.

1. The read ahead options are applicable only when one, and only one, of the MQOO_BROWSE, MQOO_INPUT_SHARED and MQOO_INPUT_EXCLUSIVE options are also specified. An error is not thrown if a read ahead options are specified with the MQOO_ INQUIRE or MQOO_SET options.
2. Read ahead is not enabled when requested if the options used on the first MQGET call are not supported for use with read ahead. Also, read ahead is disabled when the client is connecting to a queue manager that does not support read ahead.
3. If the application is not running as a IBM WebSphere MQ client, read ahead options are ignored.

Cluster queues

The following notes apply to the use of cluster queues.

1. When a cluster queue is opened for the first time, and the local queue manager is not a full repository queue manager, the local queue manager obtains information about the cluster queue from a full repository queue manager. When the network is busy, it can take several seconds for the local queue manager to receive the needed information from the repository queue manager. As a result, the application issuing the MQOPEN call might have to wait for up to 10 seconds before control returns from the MQOPEN call. If the local queue manager does not receive the needed information about the cluster queue within this time, the call fails with reason code MQRC_CLUSTER_RESOLUTION_ERROR.

2. When a cluster queue is opened and there are multiple instances of the queue in the cluster, the instance opened depends on the options specified on the MQOPEN call:
   - If the options specified include any of the following:
     – MQOO_BROWSE
     – MQOO_INPUT_AS_Q_DEF
     – MQOO_INPUT_EXCLUSIVE
     – MQOO_INPUT_SHARED
     – MQOO_SET

     the instance of the cluster queue opened must be the local instance. If there is no local instance of the queue, the MQOPEN call fails.
   - If the options specified include none of the options described previously, but include one or both of the following:
     – MQOO_INQUIRE
     – MQOO_OUTPUT

     the instance opened is the local instance if there is one, and a remote instance otherwise (if using the CLWLUSEQ defaults). The instance chosen by the queue manager can, however, be altered by a cluster workload exit (if there is one).
3. If there is a subscription for the queue, but it is not acknowledged by a full repository, the object is not present in the cluster and the call fails with reason code MQRC_OBJECT_NAME.

For more information about cluster queues, see Cluster queues.

Distribution lists

The following notes apply to the use of distribution lists.

Distribution lists are supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus IBM WebSphere MQ MQI clients connected to these systems.

1. Fields in the MQOD structure must be set as follows when opening a distribution list:
   - *Version* must be MQOD_VERSION_2 or greater.
   - *ObjectType* must be MQOT_Q.
   - *ObjectName* must be blank or the null string.
   - *ObjectQMgrName* must be blank or the null string.
   - *RecsPresent* must be greater than zero.
   - One of *ObjectRecOffset* and *ObjectRecPtr* must be zero and the other nonzero.
   - No more than one of *ResponseRecOffset* and *ResponseRecPtr* can be nonzero.
   - There must be *RecsPresent* object records, addressed by either *ObjectRecOffset* or *ObjectRecPtr*. The object records must be set to the names of the destination queues to be opened.
   - If one of *ResponseRecOffset* and *ResponseRecPtr* is nonzero, there must be *RecsPresent* response records present. They are set by the queue manager if the call completes with reason code MQRC_MULTIPLE_REASONS.

   A version-2 MQOD can also be used to open a single queue that is not in a distribution list, by ensuring that *RecsPresent* is zero.
2. Only the following open options are valid in the *Options* parameter:
   - MQOO_OUTPUT
   - MQOO_PASS_*_CONTEXT
   - MQOO_SET_*_CONTEXT
   - MQOO_ALTERNATE_USER_AUTHORITY

- MQOO_FAIL_IF_QUIESCING

3. The destination queues in the distribution list can be local, alias, or remote queues, but they cannot be model queues. If a model queue is specified, that queue fails to open, with reason code MQRC_Q_TYPE_ERROR. However, this does not prevent other queues in the list being opened successfully.
4. The completion code and reason code parameters are set as follows:
   - If the open operations for the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case.

     For example, if every open succeeds, the completion code is set to MQCC_OK and the reason code is set to MQRC_NONE; if every open fails because none of the queues exists, the parameters are set to MQCC_FAILED and MQRC_UNKNOWN_OBJECT_NAME.
   - If the open operations for the queues in the distribution list do not all succeed or fail in the same way:
     - The completion code parameter is set to MQCC_WARNING if at least one open succeeded, and to MQCC_FAILED if all failed.
     - The reason code parameter is set to MQRC_MULTIPLE_REASONS.
     - The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.
5. When a distribution list has been opened successfully, the handle *Hobj* returned by the call can be used on subsequent MQPUT calls to put messages to queues in the distribution list, and on an MQCLOSE call to relinquish access to the distribution list. The only valid close option for a distribution list is MQCO_NONE.

   The MQPUT1 call can also be used to put a message to a distribution list; the MQOD structure defining the queues in the list is specified as a parameter on that call.
6. Each successfully opened destination in the distribution list counts as a separate handle when checking whether the application has exceeded the permitted maximum number of handles (see the *MaxHandles* queue-manager attribute). This is true even when two or more of the destinations in the distribution list resolve to the same physical queue. If the MQOPEN or MQPUT1 call for a distribution list would cause the number of handles in use by the application to exceed *MaxHandles*, the call fails with reason code MQRC_HANDLE_NOT_AVAILABLE.
7. Each destination that is opened successfully has the value of its *OpenOutputCount* attribute incremented by one. If two or more of the destinations in the distribution list resolve to the same physical queue, that queue has its *OpenOutputCount* attribute incremented by the number of destinations in the distribution list that resolve to that queue.
8. Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.
9. A distribution list can contain only one destination.

Remote queues

The following notes apply to the use of remote queues.

A remote queue can be specified in one of two ways in the *ObjDesc* parameter of this call.

- By specifying for *ObjectName* the name of a local definition of the remote queue. In this case, *ObjectQMgrName* refers to the local queue manager, and can be specified as blanks or (in the C programming language) a null string.

  The security validation performed by the local queue manager verifies that the user is authorized to open the local definition of the remote queue.

- By specifying for *ObjectName* the name of the remote queue as known to the remote queue manager. In this case, *ObjectQMgrName* is the name of the remote queue manager.

  The security validation performed by the local queue manager verifies that the user is authorized to send messages to the transmission queue resulting from the name resolution process.

In either case:

- No messages are sent by the local queue manager to the remote queue manager to check that the user is authorized to put messages on the queue.
- When a message arrives at the remote queue manager, the remote queue manager might reject it because the user originating the message is not authorized.

See the *ObjectName* and *ObjectQMgrName* fields described in "MQOD - Object descriptor" on page 1768 for more information.

Objects

Security

The following notes relate to the security aspects of using MQOPEN.

The queue manager performs security checks when an MQOPEN call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. The authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved.

If the object being opened is an alias queue which points at a topic object, the queue manager performs a security check on the alias queue name, before performing a security check for the topic as if the topic object had been used directly.

If the object being opened is a topic object, whether with *ObjectName* alone or by using the *ObjectString* (with or without a basing *ObjectName*), the queue manager performs the security check by using the resultant topic string, taken from within the topic object specified in *ObjectName*, and if required concatenating it with that provided in *ObjectString*, and then finding the closest topic object at or above that point in the topic tree to perform the security check against. This might not be the same topic object that was specified in *ObjectName*.

If the object being opened is a model queue, the queue manager performs a full security check against both the name of the model queue and the name of the dynamic queue that is created. If the resulting dynamic queue is then opened explicitly, a further resource security check is performed against the name of the dynamic queue.

Attributes

The following notes relate to attributes.

The attributes of an object can change while an application has the object open. In many cases, the application does not notice this, but for certain attributes the queue manager marks the handle as no longer valid. These attributes are:

- Any attribute that affects the name resolution of the object. This applies regardless of the open options used, and includes the following:
  - A change to the *BaseQName* attribute of an alias queue that is open.
  - A change to the *TargetType* attribute of an alias queue that is open.
  - A change to the *RemoteQName* or *RemoteQMgrName* queue attributes, for any handle that is open for this queue, or for a queue that resolves through this definition as a queue-manager alias.

  - Any change that causes a currently open handle for a remote queue to resolve to a different transmission queue, or to fail to resolve to one at all. For example, this can include:
    - A change to the *XmitQName* attribute of the local definition of a remote queue, whether the definition is being used for a queue, or for a queue-manager alias.
    - (z/OS only) A change to the value of the *IntraGroupQueuing* queue-manager attribute, or a change in the definition of the shared transmission queue (SYSTEM.QSG.TRANSMIT.QUEUE) used by the IGQ agent.

    There is one exception to this: the creation of a new transmission queue. A handle that would have resolved to this queue had it been present when the handle was opened, but instead resolved to the default transmission queue, is not made invalid.
  - A change to the *DefXmitQName* queue-manager attribute. In this case all open handles that resolved to the previously named queue (that resolved to it only because it was the default transmission queue) are marked as invalid. Handles that resolved to this queue for other reasons are not affected.
- The *Shareability* queue attribute, if there are two or more handles that are currently providing MQOO_INPUT_SHARED access for this queue, or for a queue that resolves to this queue. If so, *all* handles that are open for this queue, or for a queue that resolves to this queue, are marked as invalid, regardless of the open options.

  On z/OS, the handles previously described are marked as invalid if one or more handles is currently providing MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE access to the queue.
- The *Usage* queue attribute, for all handles that are open for this queue, or for a queue that resolves to this queue, regardless of the open options.

When a handle is marked as invalid, all subsequent calls (other than MQCLOSE) using this handle fail with reason code MQRC_OBJECT_CHANGED. The application must issue an MQCLOSE call (using the original handle) and then reopen the queue. Any uncommitted updates against the old handle from previous successful calls can still be committed or backed out, as required by the application logic.

If changing an attribute causes this to happen, use a special force version of the call.

C invocation

```
MQOPEN (Hconn, &ObjDesc, Options, &Hobj, &CompCode,
       &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQOD     ObjDesc;   /* Object descriptor */
MQLONG   Options;   /* Options that control the action of MQOPEN */
MQHOBJ   Hobj;      /* Object handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQOPEN' USING HCONN, OBJDESC, OPTIONS, HOBJ, COMPCODE, REASON
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Object descriptor
 01  OBJDESC.
     COPY CMQODV.
**   Options that control the action of MQOPEN
 01  OPTIONS   PIC S9(9) BINARY.
**   Object handle
 01  HOBJ      PIC S9(9) BINARY.
```

```
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn     fixed bin(31);  /* Connection handle */
dcl ObjDesc   like MQOD;      /* Object descriptor */
dcl Options   fixed bin(31);  /* Options that control the action of
                                 MQOPEN */
dcl Hobj      fixed bin(31);  /* Object handle */
dcl CompCode  fixed bin(31);  /* Completion code */
dcl Reason    fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQOPEN,(HCONN,OBJDESC,OPTIONS,HOBJ,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS      F  Connection handle
OBJDESC   CMQODA  ,  Object descriptor
OPTIONS   DS      F  Options that control the action of MQOPEN
HOBJ      DS      F  Object handle
COMPCODE  DS      F  Completion code
REASON    DS      F  Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQOPEN Hconn, ObjDesc, Options, Hobj, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim ObjDesc  As MQOD 'Object descriptor'
Dim Options  As Long 'Options that control the action of MQOPEN'
Dim Hobj     As Long 'Object handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQPUT - Put message:

The MQPUT call puts a message on a queue or distribution list, or to a topic. The queue, distribution list, or topic must already be open.

Syntax

MQPUT (*Hconn*, *Hobj*, *MsgDesc*, *PutMsgOpts*, *BufferLength*, *Buffer*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj
Type: MQHOBJ - input

This handle represents the queue to which the message is added, or the topic to which the message is published. The value of *Hobj* was returned by a previous MQOPEN call that specified the MQOO_OUTPUT option.

MsgDesc
Type: MQMD - input/output

This structure describes the attributes of the message being sent, and receives information about the message after the put request is complete. See "MQMD - Message descriptor" on page 1705 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure to specify values for the fields that exist in the version-2 MQMD but not the version-1. The *Format* field in the MQMD must be set to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. See "MQMDE - Message descriptor extension" on page 1758 for more details.

The application does not need to provide an MQMD structure if a valid message handle is supplied in the *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure. If nothing is provided in one of these fields, the descriptor of the message is taken from the descriptor associated with the message handles.

If you use, or plan to use, API exits then we recommend that you explicitly supply an MQMD structure and do not use the message descriptors associated with the message handles. This is because the API Exit associated with MQPUT or MQPUT1 call is unable to ascertain which MQMD values are used by the queue manager to complete the MQPUT or MQPUT1 request.

PutMsgOpts
Type: MQPMO - input/output

See "MQPMO - Put-message options" on page 1791 for details.

BufferLength
Type: MQLONG - input

The length of the message in *Buffer*. Zero is valid, and indicates that the message contains no application data. The upper limit for *BufferLength* depends on various factors:

- If the destination is a local queue or resolves to a local queue, the upper limit depends on whether:
  - The local queue manager supports segmentation.
  - The sending application specifies the flag that allows the queue manager to segment the message. This flag is MQMF_SEGMENTATION_ALLOWED, and can be specified either in a version-2 MQMD, or in an MQMDE used with a version-1 MQMD.

  If both of these conditions are satisfied, *BufferLength* cannot exceed 999 999 999 minus the value of the *Offset* field in MQMD. The longest logical message that can be put is therefore 999 999 999 bytes (when *Offset* is zero). However, resource constraints imposed by the operating system or environment in which the application is running might result in a lower limit.

  If one or both of the above conditions is not satisfied, *BufferLength* cannot exceed the smaller of the queue's *MaxMsgLength* attribute and queue-manager's *MaxMsgLength* attribute.
- If the destination is a remote queue or resolves to a remote queue, the conditions for local queues apply, *but at each queue manager through which the message must pass in order to reach the destination queue*; in particular:
  1. The local transmission queue used to store the message temporarily at the local queue manager
  2. Intermediate transmission queues (if any) used to store the message at queue managers on the route between the local and destination queue managers

3. The destination queue at the destination queue manager

The longest message that can be put is therefore governed by the most restrictive of these queues and queue managers.

When a message is on a transmission queue, additional information resides with the message data, and this reduces the amount of application data that can be carried. In this situation, subtract MQ_MSG_HEADER_LENGTH bytes from the *MaxMsgLength* values of the transmission queues when determining the limit for *BufferLength*.

Note: Only failure to comply with condition 1 can be diagnosed synchronously (with reason code MQRC_MSG_TOO_BIG_FOR_Q or MQRC_MSG_TOO_BIG_FOR_Q_MGR) when the message is put. If conditions 2 or 3 are not satisfied, the message is redirected to a dead-letter (undelivered-message) queue, either at an intermediate queue manager or at the destination queue manager. If this happens, a report message is generated if one was requested by the sender.

Buffer

Type: MQBYTExBufferLength - input

This is a buffer containing the application data to be sent. The buffer must be aligned on a boundary appropriate to the nature of the data in the message. 4-byte alignment is suitable for most messages (including messages containing WebSphere MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *Buffer* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values appropriate to the data; this enables the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All the other parameters on the MQPUT call must be in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE).

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_INCONSISTENT_PERSISTENCE
(2185, X'889') Inconsistent persistence specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_PRIORITY_EXCEEDS_MAXIMUM
(2049, X'801') Message Priority exceeds maximum value supported.

MQRC_UNKNOWN_REPORT_OPTION
(2104, X'838') Report option(s) in message descriptor not recognized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_TARGTYPE_CHANGED
(2480, X'09B0') Subscription target type has changed from queue to topic object.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CALL_INTERRUPTED
(2549, X'9F5') MQPUT or MQCMIT was interrupted and reconnection processing cannot reestablish a definite outcome.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CFGR_ERROR
(2416, X'970') PCF group parameter structure MQCFGR in the message data is not valid.

MQRC_CFH_ERROR
(2235, X'8BB') PCF header structure not valid.

MQRC_CFIF_ERROR
(2414, X'96E') PCF integer filter parameter structure in the message data is not valid.

MQRC_CFIL_ERROR
(2236, X'8BC') PCF integer list parameter structure or PCIF*64 integer list parameter structure not valid.

MQRC_CFIN_ERROR
(2237, X'8BD') PCF integer parameter structure or PCIF*64 integer parameter structure not valid.

MQRC_CFSF_ERROR
(2415, X'96F') PCF string filter parameter structure in the message data is not valid.

MQRC_CFSL_ERROR
(2238, X'8BE') PCF string list parameter structure not valid.

MQRC_CFST_ERROR
(2239, X'8BF') PCF string parameter structure not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'8DD') Cluster resource error.

MQRC_COD_NOT_VALID_FOR_XCF_Q
(2106, X'83A') COD report option not valid for XCF queue.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CONTENT_ERROR
2554 (X'09FA') Message content could not be parsed to determine whether the message should be delivered to a subscriber with an extended message selector.

MQRC_CONTEXT_HANDLE_ERROR
(2097, X'831') Queue handle referred to does not save context.

MQRC_CONTEXT_NOT_AVAILABLE
(2098, X'832') Context not available for queue handle referred to.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DH_ERROR
(2135, X'857') Distribution header structure not valid.

MQRC_DLH_ERROR
(2141, X'85D') Dead letter header structure not valid.

MQRC_EPH_ERROR
(2420, X'974') Embedded PCF structure not valid.

MQRC_EXPIRY_ERROR
(2013, X'7DD') Expiry time not valid.

MQRC_FEEDBACK_ERROR
(2014, X'7DE') Feedback code not valid.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GROUP_ID_ERROR
(2258, X'8D2') Group identifier not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HEADER_ERROR
(2142, X'85E') MQ header structure not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_IIH_ERROR
(2148, X'864') IMS information header structure not valid.

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_INCONSISTENT_PERSISTENCE
(2185, X'889') Inconsistent persistence specification.

MQRC_INCONSISTENT_UOW
(2245, X'8C5') Inconsistent unit-of-work specification.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MDE_ERROR
(2248, X'8C8') Message descriptor extension not valid.

MQRC_MISSING_REPLY_TO_Q
(2027, X'7EB') Missing reply-to queue or MQPMO_SUPPRESS_REPLYTO was used

MQRC_MISSING_WIH
(2332, X'91C') Message data does not begin with MQWIH.

MQRC_MSG_FLAGS_ERROR
(2249, X'8C9') Message flags not valid.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
(2031, X'7EF') Message length greater than maximum for queue manager.

MQRC_MSG_TYPE_ERROR
(2029, X'7ED') Message type in message descriptor not valid.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_NO_DESTINATIONS_AVAILABLE
(2270, X'8DE') No destination queues available.

MQRC_NOT_OPEN_FOR_OUTPUT
(2039, X'7F7') Queue not open for output.

MQRC_NOT_OPEN_FOR_PASS_ALL
(2093, X'82D') Queue not open for pass all context.

MQRC_NOT_OPEN_FOR_PASS_IDENT
(2094, X'82E') Queue not open for pass identity context.

MQRC_NOT_OPEN_FOR_SET_ALL
(2095, X'82F') Queue not open for set all context.

MQRC_NOT_OPEN_FOR_SET_IDENT
(2096, X'830') Queue not open for set identity context.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OFFSET_ERROR
(2251, X'8CB') Message segment offset not valid.

MQRC_OPEN_FAILED
(2137, X'859') Object not opened successfully.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_ORIGINAL_LENGTH_ERROR
(2252, X'8CC') Original length not valid.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_PAGESET_FULL
(2192, X'890') External storage medium is full.

MQRC_PCF_ERROR
(2149, X'865') PCF structures not valid.

MQRC_PERSISTENCE_ERROR
(2047, X'7FF') Persistence not valid.

MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Queue does not support persistent messages.

MQRC_PMO_ERROR
(2173, X'87D') Put-message options structure not valid.

MQRC_PMO_RECORD_FLAGS_ERROR
(2158, X'86E') Put message record flags not valid.

MQRC_PRIORITY_ERROR
(2050, X'802') Message priority not valid.

MQRC_PUBLICATION_FAILURE
(2502, X'9C6') The publication has not been delivered to any of the subscribers.

MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue, for the queue to which this queue resolves, or the topic.

MQRC_PUT_MSG_RECORDS_ERROR
(2159, X'86F') Put message records not valid.

MQRC_PUT_NOT_RETAINED
(2479, X'09AF') Publication could not be retained

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.

MQRC_RECONNECT_FAILED
(2548, X'9F4') After reconnecting, an error occurred reinstating the handles for a reconnectable connection.

MQRC_RECS_PRESENT_ERROR
(2154, X'86A') Number of records present not valid.

MQRC_REPORT_OPTIONS_ERROR
(2061, X'80D') Report options in message descriptor not valid.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.

MQRC_RFH_ERROR
(2334, X'91E') MQRFH or MQRFH2 structure not valid.

MQRC_RMH_ERROR
(2220, X'8AC') Reference message header structure not valid.

MQRC_SEGMENT_LENGTH_ZERO
(2253, X'8CD') Length of data in message segment is zero.

MQRC_SEGMENTS_NOT_SUPPORTED
(2365, X'93D') Segments not supported.

MQRC_SELECTION_NOT_AVAILABLE
2551 (X'09F7') A possible subscriber for the publication exists, but the queue manager cannot check whether to send the publication to the subscriber.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_CLASS_ERROR
(2105, X'839') Storage class error.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Syncpoint support not available.

MQRC_TM_ERROR
(2265, X'8D9') Trigger message structure not valid.

MQRC_TMC_ERROR
(2191, X'88F') Character trigger message structure not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WIH_ERROR
(2333, X'91D') MQWIH structure not valid.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

MQRC_XQH_ERROR
(2260, X'8D4') Transmission queue header structure not valid.

For detailed information about these codes, see Reason codes.

Topic usage notes

1. The following notes apply to the use of topics:
   a. When using MQPUT to publish messages on a topic, where one or more subscribers to that topic cannot be given the publication due to a problem with their subscriber queue (for example it is full), the Reason code returned to the MQPUT call and the delivery behavior is dependent on the setting of the PMSGDLV or NPMSGDLV attributes on the TOPIC. Note delivery of a publication to the dead letter queue when MQRO_DEAD_LETTER_Q is specified, or discarding the message

when MQRO_DISCARD_MSG is specified, is considered as a successful delivery of the message. If none of the publications are delivered, the MQPUT returns with MQRC_PUBLICATION_FAILURE. This can happen in the following cases:

- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALL and any subscription (durable or not) has a queue which cannot receive the publication.
- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLDUR and a durable subscription has a queue which cannot receive the publication.

The MQPUT can return with MQRC_NONE even though publications could not be delivered to some subscribers in the following cases:

- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLAVAIL and any subscription, durable or not, has a queue which cannot receive the publication.
- A message is published to a TOPIC with PMSGDLV or NPMSGDLV (depending on the persistence of the message) set to ALLDUR and a non-durable subscription has a queue which cannot receive the publication.

You can use the USEDLQ topic attribute to determine whether the dead-letter queue is used when publication messages cannot be delivered to their correct subscriber queue. For more information about the use of USEDLQ, see DEFINE TOPIC.

b. If there are no subscribers to the topic being used, the message published is not sent to any queue and is discarded. It does not matter whether the message is persistent or non-persistent, or whether it has unlimited expiry or has an expiry time, it is still discarded if there are no subscribers. The exception to this is if the message is to be retained, in which case, although it is not sent to any subscribers' queues, it is stored against the topic to be delivered to any new subscriptions or to any subscribers that ask for retained publications using MQSUBRQ.

MQPUT and MQPUT1

You can use both the MQPUT and MQPUT1 calls to put messages on a queue; which call to use depends on the circumstances

- Use the MQPUT call to place multiple messages on the *same* queue.

  An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
- Use the MQPUT1 call to put only *one* message on a queue.

  This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, minimizing the number of calls that must be issued.

Destination Queues

The following notes apply to the use of destination queues:

1. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if the conditions detailed are satisfied. Some conditions apply to both local and remote destination queues; other conditions apply only to remote destination queues.

   Conditions that apply to local and remote destination queues

   - All the MQPUT calls are within the same unit of work, or none of them is within a unit of work.

     Be aware that when messages are put onto a particular queue within a single unit of work, messages from other applications might be interspersed with the sequence of messages on the queue.
   - All the MQPUT calls are made using the same object handle *Hobj*.

In some environments, message sequence is also preserved when different object handles are used, if the calls are made from the same application. The meaning of *same application* is determined by the environment:

  - On z/OS, the application is:
    - For CICS, the CICS task
    - For IMS, the task
    - For z/OS batch, the task
  - On IBM i, the application is the job.
  - On Windows and UNIX systems, the application is the thread.
- The messages all have the same priority.
- The messages are not put to a cluster queue with MQOO_BIND_NOT_FIXED specified (or with MQOO_BIND_AS_Q_DEF in effect when the DefBind queue attribute has the value MQBND_BIND_NOT_FIXED).

Additional conditions that apply to remote destination queues

- There is only one path from the sending queue manager to the destination queue manager.

  If some messages in the sequence might go on a different path (for example, because of reconfiguration, traffic balancing, or path selection based on message size), the order of the messages at the destination queue manager cannot be guaranteed.
- Messages are not placed temporarily on dead-letter queues at the sending, intermediate, or destination queue managers.

  If one or more of the messages is put temporarily on a dead-letter queue (for example, because a transmission queue or the destination queue is temporarily full), the messages can arrive on the destination queue out of sequence.
- The messages are either all persistent or all nonpersistent.

  If a channel on the route between the sending and destination queue managers has its *NonPersistentMsgSpeed* attribute set to MQNPMS_FAST, nonpersistent messages can jump ahead of persistent messages, resulting in the order of persistent messages relative to nonpersistent messages not being preserved. However, the order of persistent messages relative to each other, and of nonpersistent messages relative to each other, is preserved.

If these conditions are not satisfied, you can use message groups to preserve message order, but this requires both the sending and receiving applications to use the message-grouping support. For more information about message groups, see:

- MQMD - MsgFlags field
- MQPMO_LOGICAL_ORDER
- MQGMO_LOGICAL_ORDER

Distribution Lists

The following notes apply to the use of distribution lists.

Distribution lists are supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

1. You can put messages to a distribution list using either a version-1 or a version-2 MQPMO. If you use a version-1 MQPMO (or a version-2 MQPMO with *RecsPresent* equal to zero), the application can provide no put message records or response records. You cannot identify the queues that encounter errors if the message is sent successfully to some queues in the distribution list and not others.

   If the application provides put message records or response records, set the *Version* field to MQPMO_VERSION_2.

   You can also use a version-2 MQPMO to send messages to a single queue that is not in a distribution list, by ensuring that *RecsPresent* is zero.

2. The completion code and reason code parameters are set as follows:
   - If the puts to the queues in the distribution list all succeed or fail in the same way, the completion code and reason code parameters are set to describe the common result. The MQRR response records (if provided by the application) are not set in this case.

     For example, if every put succeeds, the completion code and reason code are set to MQCC_OK and MQRC_NONE; if every put fails because all the queues are inhibited for puts, the parameters are set to MQCC_FAILED and MQRC_PUT_INHIBITED.
   - If the puts to the queues in the distribution list do not all succeed or fail in the same way:
     – The completion code parameter is set to MQCC_WARNING if at least one put succeeded, and to MQCC_FAILED if all failed.
     – The reason code parameter is set to MQRC_MULTIPLE_REASONS.
     – The response records (if provided by the application) are set to the individual completion codes and reason codes for the queues in the distribution list.

     If the put to a destination fails because the open for that destination failed, the fields in the response record are set to MQCC_FAILED and MQRC_OPEN_FAILED; that destination is included in *InvalidDestCount*.
3. If a destination in the distribution list resolves to a local queue, the message is placed on that queue in normal form (that is, not as a distribution-list message). If more than one destination resolves to the same local queue, one message is placed on the queue for each such destination.

   If a destination in the distribution list resolves to a remote queue, a message is placed on the appropriate transmission queue. Where several destinations resolve to the same transmission queue, a single distribution-list message containing those destinations can be placed on the transmission queue, even if those destinations were not adjacent in the list of destinations provided by the application. However, this can be done only if the transmission queue supports distribution-list messages (see DistLists).

   If the transmission queue does not support distribution lists, one copy of the message in normal form is placed on the transmission queue for each destination that uses that transmission queue.

   If a distribution list with the application message data is too large for a transmission queue, the distribution list message is split into smaller distribution-list messages, each containing fewer destinations. If the application message data only just fits on the queue, distribution-list messages cannot be used at all, and the queue manager generates one copy of the message in normal form for each destination that uses that transmission queue.

   If different destinations have different message priority or message persistence (this can occur when the application specifies MQPRI_PRIORITY_AS_Q_DEF or MQPER_PERSISTENCE_AS_Q_DEF), the messages are not held in the same distribution-list message. Instead, the queue manager generates as many distribution-list messages as are necessary to accommodate the differing priority and persistence values.
4. A put to a distribution list can result in:
   - A single distribution-list message, or
   - A number of smaller distribution-list messages, or
   - A mixture of distribution list messages and normal messages, or
   - Normal messages only.

   Which of the above occurs depends on whether:
   - The destinations in the list are local, remote, or a mixture.
   - The destinations have the same message priority and message persistence.
   - The transmission queues can hold distribution-list messages.
   - The transmission queues' maximum message lengths are large enough to accommodate the message in distribution-list form.

   However, regardless of which of the above occurs, each *physical* message resulting (that is, each normal message or distribution-list message resulting from the put) counts as only *one* message when:

- Checking whether the application has exceeded the permitted maximum number of messages in a unit of work (see the *MaxUncommittedMsgs* queue-manager attribute).
- Checking whether the triggering conditions are satisfied.
- Incrementing queue depths and checking whether the queues' maximum queue depth would be exceeded.

5. Any change to the queue definitions that would have caused a handle to become invalid had the queues been opened individually (for example, a change in the resolution path), does not cause the distribution-list handle to become invalid. However, it does result in a failure for that particular queue when the distribution-list handle is used on a subsequent MQPUT call.

Headers

If a message is put with one or more WebSphere MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. If the queue manager detects an error, the call fails with an appropriate reason code. The checks performed vary according to the particular structures that are present:

- Checks are performed only if a version-2 or later MQMD is used on the MQPUT or MQPUT1 call. Checks are not performed if a version-1 MQMD is used, even if an MQMDE is present at the start of the message data.
- Structures that are not supported by the local queue manager, and structures following the first MQDLH in the message, are not validated.
- The MQDH and MQMDE structures are validated completely by the queue manager.
- Other structures are validated partially by the queue manager (not all fields are checked).

General checks performed by the queue manager include the following:

- The *StrucId* field must be valid.
- The *Version* field must be valid.
- The *StrucLength* field must specify a value that is large enough to include the structure plus any variable-length data that forms part of the structure.
- The *CodedCharSetId* field must not be zero, or a negative value that is not valid (MQCCSI_DEFAULT, MQCCSI_EMBEDDED, MQCCSI_Q_MGR, and MQCCSI_UNDEFINED are *not* valid in most WebSphere MQ header structures).
- The *BufferLength* parameter of the call must specify a value that is large enough to include the structure (the structure must not extend beyond the end of the message).

In addition to general checks on structures, the following conditions must be satisfied:

- The sum of the lengths of the structures in a PCF message must equal the length specified by the *BufferLength* parameter on the MQPUT or MQPUT1 call. A PCF message is a message that has a format name of MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF.
- A WebSphere MQ structure must not be truncated, except in the following situations where truncated structures are permitted:
  - Messages that are report messages.
  - PCF messages.
  - Messages containing an MQDLH structure. (Structures *following* the first MQDLH can be truncated; structures preceding the MQDLH cannot.)
- A WebSphere MQ structure must not be split over two or more segments; the structure must be contained entirely within one segment.

Buffer

For the Visual Basic programming language, the following points apply:

- If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_BUFFER_LENGTH_ERROR.
- The *Buffer* parameter is declared as being of type String. If the data to be placed on the queue is not of type String, use the MQPUTAny call in place of MQPUT.

  The MQPUTAny call has the same parameters as the MQPUT call, except that the *Buffer* parameter is declared as being of type Any, allowing any type of data to be placed on the queue. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.

C invocation

```
MQPUT (Hconn, Hobj, &MsgDesc, &PutMsgOpts, BufferLength, Buffer,
      &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQHOBJ   Hobj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Options that control the action of MQPUT */
MQLONG   BufferLength;  /* Length of the message in Buffer */
MQBYTE   Buffer[n];     /* Message data */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQPUT' USING HCONN, HOBJ, MSGDESC, PUTMSGOPTS, BUFFERLENGTH,
                        BUFFER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Object handle
 01  HOBJ          PIC S9(9) BINARY.
**   Message descriptor
 01  MSGDESC.
     COPY CMQMDV.
**   Options that control the action of MQPUT
 01  PUTMSGOPTS.
     COPY CMQPMOV.
**   Length of the message in BUFFER
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Message data
 01  BUFFER        PIC X(n).
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer,
           CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl Hobj          fixed bin(31);  /* Object handle */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl PutMsgOpts    like MQPMO;     /* Options that control the action of
                                     MQPUT */
dcl BufferLength  fixed bin(31);  /* Length of the message in Buffer */
dcl Buffer        char(n);        /* Message data */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQPUT,(HCONN,HOBJ,MSGDESC,PUTMSGOPTS,BUFFERLENGTH, X
               BUFFER,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN         DS       F      Connection handle
HOBJ          DS       F      Object handle
MSGDESC       CMQMDA   ,      Message descriptor
PUTMSGOPTS    CMQPMOA  ,      Options that control the action of MQPUT
BUFFERLENGTH  DS       F      Length of the message in BUFFER
BUFFER        DS       CL(n)  Message data
COMPCODE      DS       F      Completion code
REASON        DS       F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQPUT Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode,
      Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim Hobj         As Long   'Object handle'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim PutMsgOpts   As MQPMO  'Options that control the action of MQPUT'
Dim BufferLength As Long   'Length of the message in Buffer'
Dim Buffer       As String 'Message data'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

MQPUT1 - Put one message:

The MQPUT1 call puts one message on a queue, or distribution list, or to a topic.

The queue, distribution list, or topic does not need to be open.

Syntax

MQPUT1 (*Hconn*, *ObjDesc*, *MsgDesc*, *PutMsgOpts*, *BufferLength*, *Buffer*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

ObjDesc

Type: MQOD - input/output

This is a structure that identifies the queue to which the message is added, or the topic to which the message is published. See “MQOD - Object descriptor” on page 1768 for details.

If the structure is a queue, the user must be authorized to open the queue for output. The queue must **not** be a model queue.

MsgDesc
Type: MQMD - input/output

This structure describes the attributes of the message being sent, and receives feedback information after the put request is complete. See "MQMD - Message descriptor" on page 1705 for details.

If the application provides a version-1 MQMD, the message data can be prefixed with an MQMDE structure to specify values for the fields that exist in the version-2 MQMD but not the version-1. Set the *Format* field in the MQMD to MQFMT_MD_EXTENSION to indicate that an MQMDE is present. See "MQMDE - Message descriptor extension" on page 1758 for more details.

The application does not need to provide an MQMD structure if a valid message handle is supplied in the *MsgHandle* field of the MQGMO structure or in the *OriginalMsgHandle* or *NewMsgHandle* fields of the MQPMO structure. If nothing is provided in one of these fields, the descriptor of the message is taken from the descriptor associated with the message handles.

PutMsgOpts
Type: MQPMO - input/output

See "MQPMO - Put-message options" on page 1791 for details.

BufferLength
Type: MQLONG - input

The length of the message in *Buffer*. Zero is valid, and indicates that the message contains no application data. The upper limit depends on various factors; see the description of the *BufferLength* parameter of the MQPUT call for further details.

Buffer
Type: MQBYTExBufferLength - input

This is a buffer containing the application message data to be sent. Align the buffer on a boundary appropriate to the nature of the data in the message. 4-byte alignment is suitable for most messages (including messages containing WebSphere MQ header structures), but some messages might require more stringent alignment. For example, a message containing a 64-bit binary integer might require 8-byte alignment.

If *Buffer* contains character or numeric data, set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values appropriate to the data; this enables the receiver of the message to convert the data (if necessary) to the character set and encoding used by the receiver.

Note: All the other parameters on the MQPUT1 call must be in the character set and encoding of the local queue manager (given by the *CodedCharSetId* queue-manager attribute and MQENC_NATIVE).

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If the *BufferLength* parameter is zero, *Buffer* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_INCOMPLETE_GROUP
(2241, X'8C1') Message group not complete.

MQRC_INCOMPLETE_MSG
(2242, X'8C2') Logical message not complete.

MQRC_PRIORITY_EXCEEDS_MAXIMUM
(2049, X'801') Message Priority exceeds maximum value supported.

MQRC_UNKNOWN_REPORT_OPTION
(2104, X'838') Report options in message descriptor not recognized.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BACKED_OUT
(2003, X'7D3') Unit of work backed out.

MQRC_BUFFER_ERROR
(2004, X'7D4') Buffer parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_NOT_AVAILABLE
(2345, X'929') coupling facility not available.

MQRC_CF_STRUC_AUTH_FAILED
(2348, X'92C') Coupling-facility structure authorization check failed.

MQRC_CF_STRUC_ERROR
(2349, X'92D') Coupling-facility structure not valid.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CFGR_ERROR
(2416, X'970') PCF group parameter structure MQCFGR in the message data is not valid.

MQRC_CFH_ERROR
(2235, X'8BB') PCF header structure not valid.

MQRC_CFIF_ERROR
(2414, X'96E') PCF integer filter parameter structure in the message data is not valid.

MQRC_CFIL_ERROR
(2236, X'8BC') PCF integer list parameter structure or PCIF*64 integer list parameter structure not valid.

MQRC_CFIN_ERROR
(2237, X'8BD') PCF integer parameter structure or PCIF*64 integer parameter structure not valid.

MQRC_CFSF_ERROR
(2415, X'96F') PCF string filter parameter structure in the message data is not valid.

MQRC_CFSL_ERROR
(2238, X'8BE') PCF string list parameter structure not valid.

MQRC_CFST_ERROR
(2239, X'8BF') PCF string parameter structure not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CLUSTER_EXIT_ERROR
(2266, X'8DA') Cluster workload exit failed.

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
(2269, X'8DD') Cluster resource error.

MQRC_COD_NOT_VALID_FOR_XCF_Q
(2106, X'83A') COD report option not valid for XCF queue.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_QUIESCING
(2202, X'89A') Connection quiescing.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_CONTENT_ERROR
2554 (X'09FA') Message content could not be parsed to determine whether the message can be delivered to a subscriber with an extended message selector.

MQRC_CONTEXT_HANDLE_ERROR
(2097, X'831') Queue handle referred to does not save context.

MQRC_CONTEXT_NOT_AVAILABLE
(2098, X'832') Context not available for queue handle referred to.

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') DB2 subsystem not available.

MQRC_DEF_XMIT_Q_TYPE_ERROR
(2198, X'896') Default transmission queue not local.

MQRC_DEF_XMIT_Q_USAGE_ERROR
(2199, X'897') Default transmission queue usage error.

MQRC_DH_ERROR
(2135, X'857') Distribution header structure not valid.

MQRC_DLH_ERROR
(2141, X'85D') Dead letter header structure not valid.

MQRC_EPH_ERROR
(2420, X'974') Embedded PCF structure not valid.

MQRC_EXPIRY_ERROR
(2013, X'7DD') Expiry time not valid.

MQRC_FEEDBACK_ERROR
(2014, X'7DE') Feedback code not valid.

MQRC_GLOBAL_UOW_CONFLICT
(2351, X'92F') Global units of work conflict.

MQRC_GROUP_ID_ERROR
(2258, X'8D2') Group identifier not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
(2353, X'931') Handle in use for global unit of work.

MQRC_HANDLE_NOT_AVAILABLE
(2017, X'7E1') No more handles available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HEADER_ERROR
(2142, X'85E') WebSphere MQ header structure not valid.

MQRC_IIH_ERROR
(2148, X'864') IMS information header structure not valid.

MQRC_LOCAL_UOW_CONFLICT
(2352, X'930') Global unit of work conflicts with local unit of work.

MQRC_MD_ERROR
(2026, X'7EA') Message descriptor not valid.

MQRC_MDE_ERROR
(2248, X'8C8') Message descriptor extension not valid.

MQRC_MISSING_REPLY_TO_Q
(2027, X'7EB') Missing reply-to queue.

MQRC_MISSING_WIH
(2332, X'91C') Message data does not begin with MQWIH.

MQRC_MSG_FLAGS_ERROR
(2249, X'8C9') Message flags not valid.

MQRC_MSG_SEQ_NUMBER_ERROR
(2250, X'8CA') Message sequence number not valid.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
(2031, X'7EF') Message length greater than maximum for queue manager.

MQRC_MSG_TYPE_ERROR
(2029, X'7ED') Message type in message descriptor not valid.

MQRC_MULTIPLE_REASONS
(2136, X'858') Multiple reason codes returned.

MQRC_NO_DESTINATIONS_AVAILABLE
(2270, X'8DE') No destination queues available.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_OBJECT_IN_USE
(2042, X'7FA') Object already open with conflicting options.

MQRC_OBJECT_LEVEL_INCOMPATIBLE
(2360, X'938') Object level not compatible.

MQRC_OBJECT_NAME_ERROR
(2152, X'868') Object name not valid.

MQRC_OBJECT_NOT_UNIQUE
(2343, X'927') Object not unique.

MQRC_OBJECT_Q_MGR_NAME_ERROR
(2153, X'869') Object queue-manager name not valid.

MQRC_OBJECT_RECORDS_ERROR
(2155, X'86B') Object records not valid.

MQRC_OBJECT_TYPE_ERROR
(2043, X'7FB') Object type not valid.

MQRC_OD_ERROR
(2044, X'7FC') Object descriptor structure not valid.

MQRC_OFFSET_ERROR
(2251, X'8CB') Message segment offset not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_ORIGINAL_LENGTH_ERROR
(2252, X'8CC') Original length not valid.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_PAGESET_FULL
(2192, X'890') External storage medium is full.

MQRC_PCF_ERROR
(2149, X'865') PCF structures not valid.

MQRC_PERSISTENCE_ERROR
(2047, X'7FF') Persistence not valid.

MQRC_PERSISTENT_NOT_ALLOWED
(2048, X'800') Queue does not support persistent messages.

MQRC_PMO_ERROR
(2173, X'87D') Put-message options structure not valid.

MQRC_PMO_RECORD_FLAGS_ERROR
(2158, X'86E') Put message record flags not valid.

MQRC_PRIORITY_ERROR
(2050, X'802') Message priority not valid.

MQRC_PUBLICATION_FAILURE
(2502, X'9C6') The publication has not been delivered to any of the subscribers.

MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue.

MQRC_PUT_MSG_RECORDS_ERROR
(2159, X'86F') Put message records not valid.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_FULL
(2053, X'805') Queue already contains maximum number of messages.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_QUIESCING
(2161, X'871') Queue manager quiescing.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_Q_SPACE_NOT_AVAILABLE
(2056, X'808') No space available on disk for queue.

MQRC_Q_TYPE_ERROR
(2057, X'809') Queue type not valid.

MQRC_RECS_PRESENT_ERROR
(2154, X'86A') Number of records present not valid.

MQRC_REMOTE_Q_NAME_ERROR
(2184, X'888') Remote queue name not valid.

MQRC_REPORT_OPTIONS_ERROR
(2061, X'80D') Report options in message descriptor not valid.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_RESPONSE_RECORDS_ERROR
(2156, X'86C') Response records not valid.

MQRC_RFH_ERROR
(2334, X'91E') MQRFH or MQRFH2 structure not valid.

MQRC_RMH_ERROR
(2220, X'8AC') Reference message header structure not valid.

MQRC_SECURITY_ERROR
(2063, X'80F') Security error occurred.

MQRC_SEGMENT_LENGTH_ZERO
(2253, X'8CD') Length of data in message segment is zero.

MQRC_SELECTION_NOT_AVAILABLE
2551 (X'09F7') A possible subscriber for the publication exists, but the queue manager cannot check whether to send the publication to the subscriber.

MQRC_STOPPED_BY_CLUSTER_EXIT
(2188, X'88C') Call rejected by cluster workload exit.

MQRC_STORAGE_CLASS_ERROR
(2105, X'839') Storage class error.

MQRC_STORAGE_MEDIUM_FULL
(2192, X'890') External storage medium is full.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
(2024, X'7E8') No more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
(2072, X'818') Syncpoint support not available.

MQRC_TM_ERROR
(2265, X'8D9') Trigger message structure not valid.

MQRC_TMC_ERROR
(2191, X'88F') Character trigger message structure not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.

MQRC_UNKNOWN_DEF_XMIT_Q
(2197, X'895') Unknown default transmission queue.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRC_UNKNOWN_OBJECT_Q_MGR
(2086, X'826') Unknown object queue manager.

MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.

MQRC_UNKNOWN_XMIT_Q
(2196, X'894') Unknown transmission queue.

MQRC_UOW_ENLISTMENT_ERROR
(2354, X'932') Enlistment in global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
(2355, X'933') Mixture of unit-of-work calls not supported.

MQRC_UOW_NOT_AVAILABLE
(2255, X'8CF') Unit of work not available for the queue manager to use.

MQRC_WIH_ERROR
(2333, X'91D') MQWIH structure not valid.

MQRC_WRONG_CF_LEVEL
(2366, X'93E') Coupling-facility structure is wrong level.

MQRC_WRONG_MD_VERSION
(2257, X'8D1') Wrong version of MQMD supplied.

MQRC_XMIT_Q_TYPE_ERROR
(2091, X'82B') Transmission queue not local.

MQRC_XMIT_Q_USAGE_ERROR
(2092, X'82C') Transmission queue with wrong usage.

MQRC_XQH_ERROR
(2260, X'8D4') Transmission queue header structure not valid.

For detailed information about these codes, see Reason codes.

Usage notes

1. Both the MQPUT and MQPUT1 calls can be used to put messages on a queue; which call to use depends on the circumstances:
   - Use the MQPUT call to place multiple messages on the *same* queue.

     An MQOPEN call specifying the MQOO_OUTPUT option is issued first, followed by one or more MQPUT requests to add messages to the queue; finally the queue is closed with an MQCLOSE call. This gives better performance than repeated use of the MQPUT1 call.
   - Use the MQPUT1 call to put only *one* message on a queue.

     This call encapsulates the MQOPEN, MQPUT, and MQCLOSE calls into a single call, minimizing the number of calls that must be issued.
2. If an application puts a sequence of messages on the same queue without using message groups, the order of those messages is preserved if certain conditions are satisfied. However, in most environments the MQPUT1 call does not satisfy these conditions, and so does not preserve message order. The MQPUT call must be used instead in these environments. See MQPUT usage notes for details.
3. The MQPUT1 call can be used to put messages to distribution lists. For general information about this, see the usage notes for the MQOPEN and MQPUT calls.

   Distribution lists are supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

   The following differences apply when using the MQPUT1 call:

   a. If the application provides MQRR response records, they must be provided using the MQOD structure; they cannot be provided using the MQPMO structure.
   b. The reason code MQRC_OPEN_FAILED is never returned by MQPUT1 in the response records; if a queue fails to open, the response record for that queue contains the reason code resulting from the open operation.

      If an open operation for a queue succeeds with a completion code of MQCC_WARNING, the completion code and reason code in the response record for that queue are replaced by the completion and reason codes resulting from the put operation.

As with the MQOPEN and MQPUT calls, the queue manager sets the response records (if provided) only when the outcome of the call is not the same for all queues in the distribution list; this is indicated by the call completing with reason code MQRC_MULTIPLE_REASONS.

4. If the MQPUT1 call is used to put a message on a cluster queue, the call behaves as though MQOO_BIND_NOT_FIXED had been specified on the MQOPEN call.
5. If a message is put with one or more WebSphere MQ header structures at the beginning of the application message data, the queue manager performs certain checks on the header structures to verify that they are valid. For more information about this, see the usage notes for the MQPUT call.
6. If more than one of the warning situations arise (see the *CompCode* parameter), the reason code returned is the *first* one in the following list that applies:
   a. MQRC_MULTIPLE_REASONS
   b. MQRC_INCOMPLETE_MSG
   c. MQRC_INCOMPLETE_GROUP
   d. MQRC_PRIORITY_EXCEEDS_MAXIMUM or MQRC_UNKNOWN_REPORT_OPTION
7. For the Visual Basic programming language, the following points apply:
   - If the size of the *Buffer* parameter is less than the length specified by the *BufferLength* parameter, the call fails with reason code MQRC_BUFFER_LENGTH_ERROR.
   - The *Buffer* parameter is declared as being of type `String`. If the data to be placed on the queue is not of type `String`, use the MQPUT1Any call in place of MQPUT1.

     The MQPUT1Any call has the same parameters as the MQPUT1 call, except that the *Buffer* parameter is declared as being of type `Any`, allowing any type of data to be placed on the queue. However, this means that *Buffer* cannot be checked to ensure that it is at least *BufferLength* bytes in size.
8. When an MQPUT1 call is issued with MQPMO_SYNCPOINT, the default behavior changes, so that the put operation is completed asynchronously. This might cause a change in the behavior of some applications that rely on certain fields in the MQOD and MQMD structures being returned, but which now contain undefined values. An application can specify MQPMO_SYNC_RESPONSE to ensure that the put operation is performed synchronously and that all the appropriate field values are completed.

C invocation

```
MQPUT1 (Hconn, &ObjDesc, &MsgDesc, &PutMsgOpts,
       BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQOD     ObjDesc;       /* Object descriptor */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Options that control the action of MQPUT1 */
MQLONG   BufferLength;  /* Length of the message in Buffer */
MQBYTE   Buffer[n];     /* Message data */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQPUT1' USING HCONN, OBJDESC, MSGDESC, PUTMSGOPTS,
                         BUFFERLENGTH, BUFFER, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Object descriptor
 01  OBJDESC.
     COPY CMQODV.
**   Message descriptor
 01  MSGDESC.
```

```
     COPY CMQMDV.
**   Options that control the action of MQPUT1
 01  PUTMSGOPTS.
     COPY CMQPMOV.
**   Length of the message in BUFFER
 01  BUFFERLENGTH  PIC S9(9) BINARY.
**   Message data
 01  BUFFER        PIC X(n).
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

PL/I invocation

```
call MQPUT1 (Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
            CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn         fixed bin(31);  /* Connection handle */
dcl ObjDesc       like MQOD;      /* Object descriptor */
dcl MsgDesc       like MQMD;      /* Message descriptor */
dcl PutMsgOpts    like MQPMO;     /* Options that control the action of
                                     MQPUT1 */
dcl BufferLength  fixed bin(31);  /* Length of the message in Buffer */
dcl Buffer        char(n);        /* Message data */
dcl CompCode      fixed bin(31);  /* Completion code */
dcl Reason        fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
         CALL MQPUT1,(HCONN,OBJDESC,MSGDESC,PUTMSGOPTS,BUFFERLENGTH,   X
               BUFFER,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN         DS       F      Connection handle
OBJDESC       CMQODA   ,      Object descriptor
MSGDESC       CMQMDA   ,      Message descriptor
PUTMSGOPTS    CMQPMOA  ,      Options that control the action of MQPUT1
BUFFERLENGTH  DS       F      Length of the message in BUFFER
BUFFER        DS       CL(n)  Message data
COMPCODE      DS       F      Completion code
REASON        DS       F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQPUT1 Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
       CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim ObjDesc      As MQOD   'Object descriptor'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim PutMsgOpts   As MQPMO  'Options that control the action of MQPUT1'
Dim BufferLength As Long   'Length of the message in Buffer'
Dim Buffer       As String 'Message data'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

MQSET - Set object attributes:

Use the MQSET call to change the attributes of an object represented by a handle. The object must be a queue.

Syntax

MQSET (*Hconn*, *Hobj*, *SelectorCount*, *Selectors*, *IntAttrCount*, *IntAttrs*, *CharAttrLength*, *CharAttrs*, *Compcode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hobj
Type: MQHOBJ - input

This handle represents the queue object with attributes that are to be set. The handle was returned by a previous MQOPEN call that specified the MQOO_SET option.

SelectorCount
Type: MQLONG - input

This is the count of selectors that are supplied in the *Selectors* array. It is the number of attributes that are to be set. Zero is a valid value. The maximum number allowed is 256.

Selectors
Type: MQLONGxSelectorCount - input

This is an array of *SelectorCount* attribute selectors; each selector identifies an attribute (integer or character) with a value that is to be set.

Each selector must be valid for the type of queue that *Hobj* represents. Only certain MQIA_* and MQCA_* values are allowed; as listed later.

Selectors can be specified in any order. Attribute values that correspond to integer attribute selectors (MQIA_* selectors) must be specified in *IntAttrs* in the same order in which these selectors occur in *Selectors*. Attribute values that correspond to character attribute selectors (MQCA_* selectors) must be specified in *CharAttrs* in the same order in which those selectors occur. MQIA_* selectors can be interleaved with the MQCA_* selectors; only the relative order within each type is important.

You can specify the same selector more than once; if you do, the last value specified for a particular selector is the one that takes effect.

Note:

1. The integer and character attribute selectors are allocated within two different ranges; the MQIA_* selectors reside within the range MQIA_FIRST through MQIA_LAST, and the MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.

   For each range, the constants MQIA_LAST_USED and MQCA_LAST_USED define the highest value that the queue manager accepts.
2. If all the MQIA_* selectors occur first, the same element numbers can be used to address corresponding elements in the *Selectors* and *IntAttrs* arrays.

3. If the *SelectorCount* parameter is zero, *Selectors* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

The attributes that can be set are listed in the following table. No other attributes can be set using this call. For the MQCA_* attribute selectors, the constant that defines the length in bytes of the string that is required in *CharAttrs* is supplied in parentheses.

Table 218. MQSET attribute selectors for queues

Selector	Description	Note
MQCA_TRIGGER_DATA	Trigger data (MQ_TRIGGER_DATA_LENGTH).	
MQIA_DIST_LISTS	Distribution list support.	1
MQIA_INHIBIT_GET	Whether get operations are allowed.	
MQIA_INHIBIT_PUT	Whether put operations are allowed.	
MQIA_TRIGGER_CONTROL	Trigger control.	
MQIA_TRIGGER_DEPTH	Trigger depth.	
MQIA_TRIGGER_MSG_PRIORITY	Threshold message priority for triggers.	
MQIA_TRIGGER_TYPE	Trigger type.	

Note:

1. Supported only on AIX, HP-UX, IBM i, Solaris, Linux, Windows, plus WebSphere MQ MQI clients connected to these systems.

IntAttrCount

Type: MQLONG - input

This is the number of elements in the *IntAttrs* array, and must be at least the number of MQIA_* selectors in the *Selectors* parameter. Zero is a valid value if there are none.

IntAttrs

Type: MQLONGxIntAttrCount - input

This is an array of *IntAttrCount* integer attribute values. These attribute values must be in the same order as the MQIA_* selectors in the *Selectors* array.

If the *IntAttrCount* or *SelectorCount* parameter is zero, *IntAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

CharAttrLength

Type: MQLONG - input

This is the length in bytes of the *CharAttrs* parameter, and must be at least the sum of the lengths of the character attributes specified in the *Selectors* array. Zero is a valid value if there are no MQCA_* selectors in *Selectors*.

CharAttrs

Type: MQCHARxCharAttrLength - input

This is the buffer containing the character attribute values, concatenated together. The length of the buffer is given by the *CharAttrLength* parameter.

The characters attributes must be specified in the same order as the MQCA_* selectors in the *Selectors* array. The length of each character attribute is fixed (see *Selectors*). If the value to be set for an attribute contains fewer nonblank characters than the defined length of the attribute, pad the value in *CharAttrs* to the right with blanks to make the attribute value match the defined length of the attribute.

If the *CharAttrLength* or *SelectorCount* parameter is zero, *CharAttrs* is not referred to; in this case, the parameter address passed by programs written in C or System/390 assembler might be null.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed.

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CF_STRUC_FAILED
(2373, X'945') Coupling-facility structure failed.

MQRC_CF_STRUC_IN_USE
(2346, X'92A') Coupling-facility structure in use.

MQRC_CF_STRUC_LIST_HDR_IN_USE
(2347, X'92B') Coupling-facility structure list-header in use.

MQRC_CHAR_ATTR_LENGTH_ERROR
(2006, X'7D6') Length of character attributes not valid.

MQRC_CHAR_ATTRS_ERROR
(2007, X'7D7') Character attributes string not valid.

MQRC_CICS_WAIT_FAILED
(2140, X'85C') Wait request rejected by CICS.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_NOT_AUTHORIZED
(2217, X'8A9') Not authorized for connection.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_DB2_NOT_AVAILABLE
(2342, X'926') DB2 subsystem not available.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_HOBJ_ERROR
(2019, X'7E3') Object handle not valid.

MQRC_INHIBIT_VALUE_ERROR
(2020, X'7E4') Value for inhibit-get or inhibit-put queue attribute not valid.

MQRC_INT_ATTR_COUNT_ERROR
(2021, X'7E5') Count of integer attributes not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
(2023, X'7E7') Integer attributes array not valid.

MQRC_NOT_OPEN_FOR_SET
(2040, X'7F8') Queue not open for set.

MQRC_OBJECT_CHANGED
(2041, X'7F9') Object definition changed since opened.

MQRC_OBJECT_DAMAGED
(2101, X'835') Object damaged.

MQRC_PAGESET_ERROR
(2193, X'891') Error accessing page-set data set.

MQRC_Q_DELETED
(2052, X'804') Queue has been deleted.

MQRC_Q_MGR_NAME_ERROR
(2058, X'80A') Queue manager name not valid or not known.

MQRC_Q_MGR_NOT_AVAILABLE
(2059, X'80B') Queue manager not available for connection.

MQRC_Q_MGR_STOPPING
(2162, X'872') Queue manager shutting down.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SELECTOR_COUNT_ERROR
(2065, X'811') Count of selectors not valid.

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED
(2066, X'812') Count of selectors too large.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_SUPPRESSED_BY_EXIT
(2109, X'83D') Call suppressed by exit program.

MQRC_TRIGGER_CONTROL_ERROR
(2075, X'81B') Value for trigger-control attribute not valid.

MQRC_TRIGGER_DEPTH_ERROR
(2076, X'81C') Value for trigger-depth attribute not valid.

MQRC_TRIGGER_MSG_PRIORITY_ERR
(2077, X'81D') Value for trigger-message-priority attribute not valid.

MQRC_TRIGGER_TYPE_ERROR
(2078, X'81E') Value for trigger-type attribute not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. Using this call, the application can specify an array of integer attributes, or a collection of character attribute strings, or both. If no errors occur, the attributes specified are all set simultaneously. If an error occurs (for example, if a selector is not valid, or an attempt is made to set an attribute to a value that is not valid), the call fails and no attributes are set.
2. The values of attributes can be determined using the MQINQ call; see "MQINQ - Inquire object attributes" on page 2005 for details.

   Note: Not all attributes with values that can be inquired using the MQINQ call can have their values changed using the MQSET call. For example, no process-object or queue-manager attributes can be set with this call.
3. Attribute changes are preserved across restarts of the queue manager (other than alterations to temporary dynamic queues, which do not survive restarts of the queue manager).
4. You cannot change the attributes of a model queue using the MQSET call. However, if you open a model queue using the MQOPEN call with the MQOO_SET option, you can use the MQSET call to set the attributes of the dynamic local queue that is created by the MQOPEN call.
5. If the object being set is a cluster queue, there must be a local instance of the cluster queue for the open to succeed.

For more information about object attributes, see:

- "Attributes for queues" on page 2135
- "Attributes for namelists" on page 2171
- "Attributes for process definitions" on page 2174
- "Attributes for the queue manager" on page 2096

C invocation

```
MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
MQLONG   Selectors[n];    /* Array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
MQLONG   IntAttrs[n];     /* Array of integer attributes */
MQLONG   CharAttrLength;  /* Length of character attributes buffer */
MQCHAR   CharAttrs[n];    /* Character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSET' USING HCONN, HOBJ, SELECTORCOUNT, SELECTORS-TABLE,
                   INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,
                   CHARATTRS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN            PIC S9(9) BINARY.
**   Object handle
 01  HOBJ             PIC S9(9) BINARY.
**   Count of selectors
 01  SELECTORCOUNT    PIC S9(9) BINARY.
**   Array of attribute selectors
 01  SELECTORS-TABLE.
  02  SELECTORS        PIC S9(9) BINARY OCCURS n TIMES.
**   Count of integer attributes
 01  INTATTRCOUNT     PIC S9(9) BINARY.
**   Array of integer attributes
 01  INTATTRS-TABLE.
  02  INTATTRS         PIC S9(9) BINARY OCCURS n TIMES.
**   Length of character attributes buffer
 01  CHARATTRLENGTH   PIC S9(9) BINARY.
**   Character attributes
 01  CHARATTRS        PIC X(n).
**   Completion code
 01  COMPCODE         PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON           PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount,
           IntAttrs, CharAttrLength, CharAttrs, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn           fixed bin(31);  /* Connection handle */
dcl Hobj            fixed bin(31);  /* Object handle */
dcl SelectorCount   fixed bin(31);  /* Count of selectors */
dcl Selectors(n)    fixed bin(31);  /* Array of attribute selectors */
dcl IntAttrCount    fixed bin(31);  /* Count of integer attributes */
dcl IntAttrs(n)     fixed bin(31);  /* Array of integer attributes */
dcl CharAttrLength  fixed bin(31);  /* Length of character attributes
                                       buffer */
dcl CharAttrs       char(n);        /* Character attributes */
dcl CompCode        fixed bin(31);  /* Completion code */
dcl Reason          fixed bin(31);  /* Reason code qualifying
                                       CompCode */
```

High Level Assembler invocation

```
CALL MQSET,(HCONN,HOBJ,SELECTORCOUNT,SELECTORS,INTATTRCOUNT,  X
             INTATTRS,CHARATTRLENGTH,CHARATTRS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN           DS  F      Connection handle
HOBJ            DS  F      Object handle
SELECTORCOUNT   DS  F      Count of selectors
SELECTORS       DS  (n)F   Array of attribute selectors
INTATTRCOUNT    DS  F      Count of integer attributes
INTATTRS        DS  (n)F   Array of integer attributes
CHARATTRLENGTH  DS  F      Length of character attributes buffer
CHARATTRS       DS  CL(n)  Character attributes
COMPCODE        DS  F      Completion code
REASON          DS  F      Reason code qualifying COMPCODE
```

Visual Basic invocation

```
MQSET Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long   'Connection handle'
Dim Hobj           As Long   'Object handle'
Dim SelectorCount  As Long   'Count of selectors'
Dim Selectors      As Long   'Array of attribute selectors'
Dim IntAttrCount   As Long   'Count of integer attributes'
Dim IntAttrs       As Long   'Array of integer attributes'
Dim CharAttrLength As Long   'Length of character attributes buffer'
Dim CharAttrs      As String 'Character attributes'
Dim CompCode       As Long   'Completion code'
Dim Reason         As Long   'Reason code qualifying CompCode'
```

MQSETMP - Set message property:

Use the MQSET call to set or modify a property of a message handle.

Syntax

MQSETMP (*Hconn*, *Hmsg*, *SetPropOpts*, *Name*, *PropDesc*, *Type*, *ValueLength*, *Value*, *Compcode*, *Reason*)

Parameters

Hconn
Type: MQHCONN - input

This handle represents the connection to the queue manager.

The value must match the connection handle that was used to create the message handle specified in the *Hmsg* parameter. If the message handle was created using MQHC_UNASSOCIATED_HCONN, a valid connection must be established on the thread setting a property of the message handle, otherwise the call fails with reason code MQRC_CONNECTION_BROKEN.

Hmsg
Type: MQHMSG - input

This is the message handle to be modified. The value was returned by a previous MQCRTMH call.

SetPropOpts
Type: MQSMPO - input

Control how message properties are set.

This structure allows applications to specify options that control how message properties are set. The structure is an input parameter on the MQSETMP call. See MQSMPO for further information.

Name
Type: MQCHARV- input

This is the name of the property to set.

See Property names and Property name restrictions for further information about the use of property names.

PropDesc
Type: MQPD - input/output

This structure is used to define the attributes of a property, including:

- what happens if the property is not supported
- what message context the property belongs to
- what messages the property is copied into as it flows

See MQPD for further information about this structure.

Type

Type: MQLONG - input

The data type of the property being set. It can be one of the following:

MQTYPE_BOOLEAN
A Boolean. *ValueLength* must be 4.

MQTYPE_BYTE_STRING
A byte string. *ValueLength* must be zero or greater.

MQTYPE_INT8
An 8-bit signed integer. *ValueLength* must be 1.

MQTYPE_INT16
A 16-bit signed integer. *ValueLength* must be 2.

MQTYPE_INT32
A 32-bit signed integer.*ValueLength* must be 4.

MQTYPE_INT64
A 64-bit signed integer.*ValueLength* must be 8.

MQTYPE_FLOAT32
A 32-bit floating-point number. *ValueLength* must be 4.

Note: this type is not supported with applications using IBM COBOL for z/OS.

MQTYPE_FLOAT64
A 64-bit floating-point number. *ValueLength* must be 8.

Note: this type is not supported with applications using IBM COBOL for z/OS.

MQTYPE_STRING
A character string. *ValueLength* must be zero or greater, or the special value MQVL_NULL_TERMINATED.

MQTYPE_NULL
The property exists but has a null value.*ValueLength* must be zero.

ValueLength

Type: MQLONG - input

The length in bytes of the property value in the *Value* parameter. Zero is valid only for null values or for strings or byte strings. Zero indicates that the property exists but that the value contains no characters or bytes.

The value must be greater than or equal to zero or the following special value if the *Type* parameter has MQTYPE_STRING set:

MQVL_NULL_TERMINATED
The value is delimited by the first null encountered in the string. The null is not included as part of the string. This value is invalid if MQTYPE_STRING is not also set.

Note: The null character used to terminate a string if MQVL_NULL_TERMINATED is set is a null from the character set of the Value.

Value

Type: MQBYTExValueLength - input

The value of the property to be set. The buffer must be aligned on a boundary appropriate to the nature of the data in the value.

In the C programming language, the parameter is declared as a pointer-to-void; the address of any type of data can be specified as the parameter.

If *ValueLength* is zero, *Value* is not referred to. In this case, the parameter address passed by programs written in C or System/390 assembler can be null.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_RFH_FORMAT_ERROR
(2421, X'0975') An MQRFH2 folder containing properties could not be parsed.

If *CompCode* is MQCC_FAILED:

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'089C') Adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
(2130, X'852') Unable to load adapter service module.

MQRC_ASID_MISMATCH
(2157, X'86D') Primary and home ASIDs differ.

MQRC_BUFFER_ERROR
(2004, X'07D4') Value parameter not valid.

MQRC_BUFFER_LENGTH_ERROR
(2005, X'07D5') Value length parameter not valid.

MQRC_CALL_IN_PROGRESS
(2219, X'08AB') MQI call entered before previous call completed.

MQRC_HMSG_ERROR
(2460, X'099C') Message handle pointer not valid.

MQRC_MSG_HANDLE_IN_USE
(2499, X'09C3') Message handle already in use.

MQRC_OPTIONS_ERROR
(2046, X'07FE') Options not valid or not consistent.

MQRC_PD_ERROR
(2482, X'09B2') Property descriptor structure not valid.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid property name.

MQRC_PROPERTY_TYPE_ERROR
(2473, X'09A9') Invalid property data type.

MQRC_PROP_NUMBER_FORMAT_ERROR
(2472, X'09A8') Number format error encountered in value data.

MQRC_SMPO_ERROR
(2463, X'099F') Set message property options structure not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'083F') Property name coded character set identifier not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQSETMP (Hconn, Hmsg, &SetPropOpts, &Name, &PropDesc, Type,
ValueLength, &Value, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;       /* Connection handle */
MQHMSG   Hmsg;        /* Message handle */
MQSMPO   SetPropOpts; /* Options that control the action of MQSETMP */
MQCHARV  Name;        /* Property name */
MQPD     PropDesc;    /* Property descriptor */
MQLONG   Type;        /* Property data type */
MQLONG   ValueLength; /* Length of property value in Value */
MQBYTE   Value[n];    /* Property value */
MQLONG   CompCode;    /* Completion code */
MQLONG   Reason;      /* Reason code qualifying CompCode */
```

COBOL invocation

```
     CALL 'MQSETMP' USING HCONN, HMSG, SETMSGOPTS, NAME, PROPDESC, TYPE,
                          VALUELENGTH, VALUE, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN     PIC S9(9) BINARY.
**   Message handle
 01  HMSG    PIC S9(18) BINARY.
**   Options that control the action of MQSETMP
 01  SETMSGOPTS.
     COPY CMQSMPOV.
**   Property name
 01  NAME
    COPY CMQCHRVV.
**   Property descriptor
 01  PROPDESC.
     COPY CMQPDV.
**   Property data type
 01  TYPE          PIC S9(9) BINARY.
**   Length of property value in VALUE
 01  VALUELENGTH   PIC S9(9) BINARY.
**   Property value
 01  VALUE         PIC X(n).
**   Completion code
 01  COMPCODE  PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON    PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSETMP (Hconn, Hmsg, SetPropOpts, Name, PropDesc, Type, ValueLength,
              Value, CompCode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn       fixed bin(31);  /* Connection handle */
dcl Hmsg        fixed bin(63);  /* Message handle */
dcl SetPropOpts like MQSMPO;    /* Options that control the action of MQSETMP */
dcl Name        like MQCHARV;   /* Property name */
dcl PropDesc    like MQPD;      /* Property descriptor */
dcl Type        fixed bin(31);  /* Property data type */
dcl ValueLength fixed bin(31);  /* Length of property value in Value */
dcl Value       char(n);        /* Property value */
dcl CompCode    fixed bin(31);  /* Completion code */
dcl Reason      fixed bin(31);  /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
         CALL MQSETMP,(HCONN,HMSG,SETMSGHOPTS,NAME,PROPDESC,TYPE,VALUELENGTH,
               VALUE,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN        DS         F     Connection handle
HMSG         DS         D     Message handle
SETMSGOPTS   CMQSMPOA   ,     Options that control the action of MQSETMP
NAME         CMQCHRVA   ,     Property name
PROPDESC     CMQPDA     ,     Property descriptor
TYPE         DS         F     Property data type
VALUELENGTH  DS         F     Length of property value in VALUE
VALUE        DS         CL(n) Property value
COMPCODE     DS         F     Completion code
REASON       DS         F     Reason code qualifying COMPCODE
```

MQSTAT - Retrieve status information:

Use the MQSTAT call to retrieve status information. The type of status information returned is determined by the Type value specified on the call.

Syntax

MQSTAT (*Hconn*, *Type*, *Stat*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Type

Type: MQLONG - input

Type of status information being requested. The > valid values are:

MQSTAT_TYPE_ASYNC_ERROR
Return information about previous asynchronous put operations.

MQSTAT_TYPE_RECONNECTION

Return information about reconnection. If the connection is reconnecting or failed to reconnect, the information describes the failure which caused the connection to begin reconnecting.

This value is only valid for client connections. For other types of connection, the call fails with reason code **MQRC_ENVIRONMENT_ERROR**

MQSTAT_TYPE_RECONNECTION_ERROR

Return information about a previous failure related to reconnect. If the connection failed to reconnect, the information describes the failure which caused reconnection to fail.

This value is only valid for client connections. For other types of connection, the call fails with reason code **MQRC_ENVIRONMENT_ERROR** .

Stat

Type: MQSTS - input/output

Status information structure. See "MQSTS - Status reporting structure" on page 1887 for details.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_API_EXIT_ERROR
(2374, X'946') API exit failed

MQRC_API_EXIT_LOAD_ERROR
(2183, X'887') Unable to load API exit.

MQRC_CALL_IN_PROGRESS
(2219, X'8AB') MQI call entered before previous call complete.

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_CONNECTION_STOPPING
(2203, X'89B') Connection shutting down.

MQRC_FUNCTION_NOT_SUPPORTED
(2298, X'8FA') The function requested is not available in the current environment.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_Q_MGR_STOPPING
(2162,X'872' - Queue manager stopping

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_STAT_TYPE_ERROR
(2430, X'97E' Error with MQSTAT type

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_STS_ERROR
(2426, X'97A') Error with MQSTS structure

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

Usage notes

1. A call to MQSTAT specifying a type of MQSTAT_TYPE_ASYNC_ERROR returns information about previous asynchronous MQPUT and MQPUT1 operations. The MQSTS structure passed back on return from the MQSTAT call contains the first recorded asynchronous warning or error information for that connection. If further errors or warnings follow the first, they do not normally alter these values. However, if an error occurs with a completion code of MQCC_WARNING, a subsequent failure with a completion code of MQCC_FAILED is returned instead.
2. If no errors have occurred since the connection was established or since the last call to MQSTAT then a CompCode of MQCC_OK and Reason of MQRC_NONE are returned in the MQSTS structure.
3. Counts of the number of asynchronous calls that have been processed under the connection handle are returned by way of three counter fields; PutSuccessCount, PutWarningCount and PutFailureCount. These counters are incremented by the queue manager each time an asynchronous operation is processed successfully, has a warning, or fails (note that for accounting purposes a put to a distribution list counts once per destination queue rather than once per distribution list). A counter is not incremented beyond the maximum positive value AMQ_LONG_MAX.
4. A successful call to MQSTAT results in any previous error information or counts being reset.
5. The behavior of MQSTAT depends on the value of the MQSTAT Type parameter you provide.
6.

   MQSTAT_TYPE_ASYNC_ERROR

   a. A call to MQSTAT specifying a type of MQSTAT_TYPE_ASYNC_ERROR returns information about previous asynchronous MQPUT and MQPUT1 operations. The MQSTS structure passed back on return from the MQSTAT call contains the first recorded asynchronous warning or error information for that connection. If further errors or warnings follow the first, they do not normally alter these values. However, if an error occurs with a completion code of MQCC_WARNING, a subsequent failure with a completion code of MQCC_FAILED is returned instead.
   b. If no errors have occurred since the connection was established or since the last call to MQSTAT then a CompCode of MQCC_OK and Reason of MQRC_NONE are returned in the MQSTS structure.
   c. Counts of the number of asynchronous calls that have been processed under the connection handle are returned by way of three counter fields; PutSuccessCount, PutWarningCount and PutFailureCount. These counters are incremented by the queue manager each time an asynchronous operation is processed successfully, has a warning, or fails (note that for accounting purposes a put to a distribution list counts once per destination queue rather than once per distribution list). A counter is not incremented beyond the maximum positive value AMQ_LONG_MAX.
   d. A successful call to MQSTAT results in any previous error information or counts being reset.

MQSTAT_TYPE_RECONNECTION

Suppose you call MQSTAT with Type set to MQSTAT_TYPE_RECONNECTION inside an event handler during reconnection. Consider these examples.

The client is attempting reconnection or failed to reconnect.
CompCode in the MQSTS structure is MQCC_FAILED and Reason might be either MQRC_CONNECTION_BROKEN or MQRC_Q_MGR_QUIESCING . ObjectType is MQOT_Q_MGR, ObjectName is the name of the queue manager, and ObjectQMgrName is blank.

The client completed reconnection successfully or was never disconnected.
CompCode in the MQSTS structure is MQCC_OK and the Reason is MQRC_NONE

Subsequent calls to MQSTAT return the same results.

MQSTAT_TYPE_RECONNECTION_ERROR

Suppose you call MQSTAT with Type set to MQSTAT_TYPE_RECONNECTION_ERROR in response to receiving MQRC_RECONNECT_FAILED to an MQI call. Consider these examples.

An authorization failure occurred when a queue was being reopened during reconnection to a different queue manager.
CompCode in the MQSTS structure is MQCC_FAILED and Reason is the reason that the reconnection failed, such as MQRC_NOT_AUTHORIZED . ObjectType is the type of object that caused the problem, such as MQOT_QUEUE, ObjectName is the name of the queue and ObjectQMgrName the name of the queue manager owning the queue.

A socket connection error occurred during reconnection.
CompCode in the MQSTS structure is MQCC_FAILED and Reason is the reason that the reconnection failed, such as MQRC_HOST_NOT_AVAILABLE . ObjectType is MQOT_Q_MGR, ObjectName is the name of the queue manager, and ObjectQMgrName is blank.

Subsequent calls to MQSTAT return the same results.

C invocation

```
MQSTAT (Hconn, StatType, &Stat, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN Hconn;  /* Connection Handle */
MQLONG StatType; /* Status type */
MQSTS Stat;       /* Status information structure */
MQLONG CompCode; /* Completion code */
MQLONG Reason;  /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSTAT' USING HCONN, STATTYPE, STAT, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
 01 HCONN  PIC S9(9) BINARY.
** Status type
 01 STATTYPE PIC S9(9) BINARY.
** Status information
 01 STAT.
 COPY CMQSTSV.
** Completion code
 01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
 01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSTAT (Hconn, StatType, Stat, Compcode, Reason);
```

Declare the parameters as follows:

```
dcl Hconn  fixed bin(31); /* Connection handle */
dcl StatType fixed bin(31); /* Status type */
dcl Stat   like MQSTS;   /* Status information structure */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason  fixed bin(31); /* Reason code qualifying CompCode */
```

System/390 Assembler invocation

```
CALL MQSTAT,(HCONN,STATTYPE,STAT,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS     F    Connection handle
STATTYPE  DS     F    Status type
STAT      CMQSTSA,    Status information structure
COMPCODE  DS     F    Completion code
REASON    DS     F    Reason code qualifying COMPCODE
```

MQSUB - Register subscription:

Use the MQSUB call to register the applications subscription to a particular topic.

Syntax

MQSUB (*Hconn*, *SubDesc*, *Hobj*, *Hsub*, *Compcode* , *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn* :

MQHC_DEF_HCONN
Default connection handle.

SubDesc

Type: MQSD - input/output

This is a structure that identifies the object in use that is being registered by the application. See "MQSD - Subscription descriptor" on page 1860 for more information.

Hobj

Type: MQHOBJ - input/output

This handle represents the access that has been established to obtain the messages sent to this subscription. These messages can either be stored on a specific queue or the queue manager can manage their storage without using a specific queue.

To use a specific queue, you must associate it with the subscription when the subscription is created. You can do this in two ways:

- By using the DEFINE SUB MQSC command and provided that command with the name of a queue object.
- By providing this handle when calling MQSUB with the MQSO_CREATE

If this handle is provided as an input parameter on the call, it must be a valid object handle returned from a previous MQOPEN call of a queue using at least one of the following options:

- MQOO_INPUT_*
- MQOO_BROWSE
- MQOO_OUTPUT (if the queue is a remote queue)

If this is not the case, the call fails with MQRC_HOBJ_ERROR. It cannot be an object handle to an alias queue that resolves to a topic object. If so, the call fails with MQRC_HOBJ_ERROR.

If the queue manager is to manage the storage of messages sent to this subscription, this should be set when you create the subscription, by using the MQSO_MANAGED option. The queue manager then returns this handle as an output parameter on the call. The handle that is returned is known as a managed handle. If MQHO_NONE is specified but MQSO_MANAGED is not specified, the call fails with MQRC_HOBJ_ERROR.

When a managed handle is returned to you by the queue manager, you can use it on an MQGET or MQCB call with or without browse options, on an MQINQ call, or on MQCLOSE. You cannot use it on MQPUT, MQSUB, MQSET; attempting to do so fails with MQRC_NOT_OPEN_FOR_OUTPUT, MQRC_HOBJ_ERROR, or MQRC_NOT_OPEN_FOR_SET.

If this subscription is being resumed using the MQSO_RESUME option in the MQSD structure, the handle can be returned to the application in this parameter by setting MQSO_MANAGED to MQHO_NONE. You can do this whether the subscription is using a managed handle or not and it can be useful to provide subscriptions created using DEFINE SUB with the handle to the subscription queue defined on that command. In the case where an administratively created subscription is being resumed, the queue opens with MQOO_INPUT_AS_Q_DEF and MQOO_BROWSE. If you need to specify other options, the application must open the subscription queue explicitly and provide the object handle on the call. If there is a problem opening the queue the call fails with MQRC_INVALID_DESTINATION. If the *Hobj* is provided, it must be equivalent to the *Hobj* in the original MQSUB call. This means if an object handle returned from an MQOPEN call is being provided, the handle must be to the same queue as previously used. If it is not the same queue, the call fails with MQRC_HOBJ_ERROR.

If this subscription is being altered using the MQSO_ALTER option in the MQSD structure, then a different *Hobj* can be provided. Any publications that have been delivered to the queue and were previously identified through this parameter stay on that queue and it is the responsibility of the application to retrieve those messages if the *Hobj* parameter now represents a different queue.

The table summarizes the use of this parameter with various subscription options:

Options	*Hobj*	**Description**
MQSO_CREATE + MQSO_MANAGED	Ignored on input	Creates a subscription with storage of messages managed by the queue manager
MQSO_CREATE	A valid object handle	Creates a subscription providing a specific queue as the destination for messages.
MQSO_RESUME	MQHO_NONE	Resumes a previously created subscription whether it was managed or not, and has the queue manager return the object handle for use by the application.
MQSO_RESUME	A valid, matching, object handle	Resumes a previously created subscription that uses a specific queue as the destination for messages and use an object handle with specific open options.

Options	*Hobj*	Description
MQSO_ALTER + MQSO_MANAGED	MQHO_NONE	Alters an existing subscription that was previously using a specific queue, so it is now a managed subscription. The class of destination (managed or not) cannot be changed.
MQSO_ALTER	A valid object handle	Alters an existing subscription, whether it was managed or not, so that it now uses a specific queue. When the MQSO_MANAGED option is not used, the queue provided can be changed, but the class of destination (managed or not) cannot be changed.

Whether it was provided or returned, *Hobj* must be specified on subsequent MQGET or MQCB calls that want to receive the publication messages sent to this subscription.

The *Hobj* handle is no longer valid when the MQCLOSE call is issued on it, or when the unit of processing that defines the scope of the handle terminates (until the application disconnects). The scope of the object handle returned is the same as that of the connection handle specified on the call. See Hconn (MQHCONN) - output for information about handle scope. An MQCLOSE of the *Hobj* handle does not affect the *Hsub* handle.

Hsub

Type: MQHOBJ - output

This handle represents the subscription that has been made. It can be used for two further operations:

- It can be used on a subsequent MQSUBRQ call to request that publications be sent when the MQSO_PUBLICATIONS_ON_REQUEST option has been used when making the subscription.
- It can be used on a subsequent MQCLOSE call to remove the subscription that has been made. The *Hsub* handle ceases to be valid when the MQCLOSE call is issued, or when the unit of processing that defines the scope of the handle terminates. The scope of the object handle returned is the same as that of the connection handle specified on the call. An MQCLOSE of the *Hsub* handle does not affect the *Hobj* handle.

This handle cannot be passed to an MQGET or MQCB call. You must use the *Hobj* parameter. You cannot use this handle on any WebSphere MQ call other than MQCLOSE or MQSUBRQ. Passing this handle to any other WebSphere MQ call results in MQRC_HOBJ_ERROR.

CompCode

Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion

MQCC_WARNING
Warning (partial completion)

MQCC_FAILED
Call failed

Reason

Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK, the reason code is as follows:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED, the reason code is one of the following:

MQRC_CLUSTER_RESOLUTION_ERROR
(2189, X'88D') Cluster name resolution failed.

MQRC_DURABILITY_NOT_ALLOWED
2436 (X'0984') An MQSUB call using the MQSO_DURABLE option failed.

MQRC_FUNCTION_NOT_SUPPORTED
2298 (X'08FA') The function requested is not available in the current environment.

MQRC_HOBJ_ERROR
2019 (X'07E3') Object handle Hobj not valid.

MQRC_IDENTITY_MISMATCH
2434 (X'0982') Subscription name matches existing subscription.

MQRC_NOT_AUTHORIZED
2035 (X'07F3') The user is not authorized to perform the operation.

MQRC_OBJECT_STRING_ERROR
2441 (X'0989') Objectstring field not valid.

MQRC_OPTIONS_ERROR
2046 (X'07FE') Options parameter or field contains options that are not valid, or a combination of options that is not valid.

MQRC_Q_MGR_QUIESCING
2161 (X'0871') Queue manager quiescing.

MQRC_RECONNECT_Q_MGR_REQD
2555 (X'09FB'X) The MQCNO_RECONNECT_Q_MGR option is required.

MQRC_RETAINED_MSG_Q_ERROR
2525 (X'09DD') Retained publications which exist for the subscribed topic string, cannot be retrieved.

MQRC_RETAINED_NOT_DELIVERED
2526 (X'09DE') The retained publications which exist for the subscribed topic string, cannot be delivered to the subscription destination queue, and cannot be delivered to the dead-letter queue.

MQRC_SD_ERROR
2424 (X'0978') Subscription descriptor (MQSD) not valid.

MQRC_SELECTION_NOT_AVAILABLE
2551 (X'09F7') The selection string does not follow the WebSphere MQ selector syntax and no extended message selection provider was available.

MQRC_SELECTION_STRING_ERROR
2519 (X'09D7') The selection string must be specified as described in the MQCHARV structure documentation.

MQRC_SELECTOR_SYNTAX_ERROR
2459 (X'099B') An MQOPEN, MQPUT1, or MQSUB call was issued but a selection string was specified which contained a syntax error.

MQRC_SUB_USER_DATA_ERROR
2431 (X'097F') SubUserData field not valid.

MQRC_SUB_NAME_ERROR
2440 (X'0988') SubName field not valid.

MQRC_SUB_ALREADY_EXISTS
2432 (X'0980') Subscription already exists.

MQRC_SUB_USER_DATA_ERROR
2431 (X'097F') SubUserData field not valid.

MQRC_TOPIC_STRING_ERROR
2425 (X'0979') Topic string is not valid.

MQRC_UNKNOWN_OBJECT_NAME
2085 (X'0825') Object identified cannot be found.

For detailed information about these codes, see Reason codes.

Usage notes

1. The subscription is made to a topic, named either using the short name of a pre-defined topic object, the full name of the topic string, or it is formed by the concatenation of two parts. See the description of *ObjectName* and *ObjectString* in "MQSD - Subscription descriptor" on page 1860.
2. The queue manager performs security checks when an MQSUB call is issued, to verify that the user identifier under which the application is running has the appropriate level of authority before access is permitted. The appropriate topic object is located in the topic hierarchy and an authority check is made on this topic object to ensure authority to subscribe is set. If the MQSO_MANAGED option is not used, an authority check is made on the destination queue to ensure that authority for output is set. If the MQSO_MANAGED option is used, no authority check is made on the managed queue for output or inquire access.
3. If you do not provide an Hobj as input, the MQSUB call allocates two handles, an object handle (Hobj) and a subscription handle (Hsub).
4. The Hobj returned on the MQSUB call when the MQSO_MANAGED option is used, can be inquired in order to find out attributes such as the Backout threshold and the Excessive backout requeue name. You can also inquire the name of the managed queue, but you must not attempt to directly open this queue.
5. Subscriptions can be grouped allowing only a single publication to be delivered to the group of subscriptions even where more than one of the group matched the publication. Subscriptions are grouped using the MQSO_GROUP_SUB option and in order to group subscriptions they must be
   - using the same named queue (that is not using the MQSO_MANAGED option) on the same queue manager - represented by the Hobj parameter on the MQSUB call
   - share the same SubCorrelId
   - be of the same SubLevel

   These attributes define the set of subscriptions considered to be in the group, and are also the attributes that cannot be altered if a subscription is grouped. Alteration of SubLevel results in MQRC_SUBLEVEL_NOT_ALTERABLE, and alteration of any of the others (which can be changed if a subscription is not grouped) results in MQRC_GROUPING_NOT_ALTERABLE.
6. Fields in the MQSD are filled in on return from an MQSUB call which uses the MQSO_RESUME option. The MQSD returned can be passed directly into an MQSUB call which uses the MQSO_ALTER option with any changes you need to make to the subscription applied to the MQSD. Some fields have special considerations as noted in the table.

MQSD output from MQSUB

Field name in MQSD	Special considerations
Access or creation options	Some of the options can be reset on return from the MQSUB call. If you then reuse the MQSD in an MQSUB call, the option you require must be explicitly set.
Durability options, Destination options, Registration Options & Wildcard options	These options are set as appropriate
Publication options	These options are set as appropriate, except for MQSO_NEW_PUBLICATIONS_ONLY which is only applicable to MQSO_CREATE.
Other options	These options are unchanged on return from an MQSUB call. They control how the API call is issued and are not stored with the subscription. They must be set as required on any subsequent MQSUB call reusing the MQSD.
ObjectName	This input only field is unchanged on return from an MQSUB call.
ObjectString	This input only field is unchanged on return from an MQSUB call. The Full topic name used is returned in the *ResObjectString* field, if a buffer is provided.
AlternateUserId and AlternateSecurityId	These input only fields are unchanged on return from an MQSUB call. They control how the API call is issued and are not stored with the subscription. They must set as required on any subsequent MQSUB call reusing the MQSD.
SubExpiry	On return from an MQSUB call using the MQSO_RESUME option, this field is set to the original expiry of the subscription and not the remaining expiry time. If you then reuse the MQSD in an MQSUB call using the MQSO_ALTER option you reset the expiry of the subscription to start counting down again.
SubName	This field is an input field on an MQSUB call and is not changed on output.
SubUserData and SelectionString	These variable length fields are returned on output from an MQSUB call using the MQSO_RESUME option, if a buffer is provided, and also a positive buffer length in *VSBufSize*. If no buffer is provided only the length is returned in the *VSLength* field of the MQCHARV. If the buffer provided is smaller than the space required to return the field, only *VSBufSize* bytes are returned in the provided buffer. If you then reuse the MQSD in an MQSUB call using the MQSO_ALTER option and a buffer is not provided but a non-zero *VSLength* is provided, if that length matches the existing length of the field, no alteration is made to the field.

MQSD output from MQSUB

Field name in MQSD	Special considerations
SubCorrelId and PubAccountingToken	If you do not use MQSO_SET_CORREL_ID, then the *SubCorrelId* is generated by the queue manager. If you do not use MQSO_SET_IDENTITY_CONTEXT, then the *PubAccountingToken* is generated by the queue manager. These fields are returned in the MQSD from an MQSUB call using the MQSO_RESUME option. If they are generated by the queue manager, the generated value is returned on an MQSUB call using the MQSO_CREATE or MQSO_ALTER option.
PubPriority, SubLevel & PubApplIdentityData	These fields are returned in the MQSD.
ResObjectString	This output only field is returned in the MQSD if a buffer is provided.

C invocation

```
MQSUB (Hconn, &SubDesc, &Hobj, &Hsub, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn;    /* Connection handle */
MQSD    SubDesc;  /* Subscription descriptor */
MQHOBJ  Hobj;     /* Object handle */
MQHOBJ  Hsub;     /* Subscription handle */
MQLONG  CompCode; /* Completion code */
MQLONG  Reason;   /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSUB' USING HCONN, SUBDESC, HOBJ, HSUB, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN    PIC S9(9) BINARY.
**   Subscription descriptor
 01  SUBDESC.
     COPY CMQSDV.
**   Object handle
 01  HOBJ     PIC S9(9) BINARY.
**   Subscription handle
 01  HSUB     PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON   PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSUB (Hconn, SubDesc, Hobj, Hsub, CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn    fixed bin(31); /* Connection handle */
dcl SubDesc  like MQSD;     /* Subscription descriptor */
dcl Hobj     fixed bin(31); /* Object handle */
dcl Hsub     fixed bin(31); /* Subscription handle */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason   fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQSUB,(HCONN,SUBDESC,HOBJ,HSUB,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN    DS      F  Connection handle
SUBDESC  CMQSDA  ,  Subscription descriptor
HOBJ     DS      F  Object handle
HSUB     DS      F  Subscription handle
COMPCODE DS      F  Completion code
REASON   DS      F  Reason code qualifying COMPCODE
```

MQSUBRQ - Subscription request:

Use the MQSUBRQ call to make a request for the retained publication, when the subscriber has been registered with MQSO_PUBLICATIONS_ON_REQUEST.

Syntax

MQSUBRQ (*Hconn*, *Hsub*, *Action*, *SubRqOpts*, *Compcode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN or MQCONNX call.

On z/OS for CICS applications, and on IBM i for applications running in compatibility mode, the MQCONN call can be omitted, and the following value specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

Hsub

Type: MQHOBJ - input

This handle represents the subscription for which an update is to be requested. The value of *Hsub* was returned from a previous MQSUB call.

Action

Type: MQLONG - input

This parameter controls the particular action that is being requested on the subscription. The following value must be specified:

MQSR_ACTION_PUBLICATION

This action requests that an update publication is sent for the specified topic. It can be used only if the subscriber specified the option MQSO_PUBLICATIONS_ON_REQUEST on the MQSUB call when it made the subscription. If the queue manager has a retained publication for the topic, this is sent to the subscriber. If not, the call fails. If an application is sent a publication which was retained, this is indicated by the MQIsRetained message property of that publication.

Since the topic in the existing subscription represented by the Hsub parameter can contain wildcards, the subscriber might receive multiple retained publications.

SubRqOpts

Type: MQSRO - input/output

These options control the action of MQSUBRQ, see "MQSRO - Subscription request options" on page 1885 for details.

If no options are required, programs written in C or S/390 assembler can specify a null parameter address instead of specifying the address of an MQSRO structure.

CompCode
Type: MQLONG - output

The completion code; it is one of the following:

MQCC_OK
Successful completion

MQCC_WARNING
Warning (partial completion)

MQCC_FAILED
Call failed

Reason
Type: MQLONG - output

The reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_FUNCTION_NOT_SUPPORTED
2298 (X'08FA') The function requested is not available in the current environment.

MQRC_NO_RETAINED_MSG
2437 (X'0985') There are no retained publications currently stored for this topic.

MQRC_OPTIONS_ERROR
2046 (X'07FE') Options parameter or field contains options that are not valid, or a combination of options that is not valid.

MQRC_Q_MGR_QUIESCING
2161 (X'0871') Queue manager quiescing.

MQRC_SRO_ERROR
2438 (X'0986') On the MQSUBRQ call, the Subscription Request Options MQSRO is not valid.

MQRC_RETAINED_MSG_Q_ERROR
2525 (X'09DD') Retained publications which exist for the subscribed topic string, cannot be retrieved.

MQRC_RETAINED_NOT_DELIVERED
2526 (X'09DE') The retained publications which exist for the subscribed topic string, cannot be delivered to the subscription destination queue, and cannot be delivered to the dead-letter queue.

For detailed information about these codes, see Reason codes.

Usage notes

The following usage notes apply to the use of the Action code MQSR_ACTION_PUBLICATION:

1. If this verb completes successfully, the retained publications matching the subscription specified have been sent to the subscription and can be received by using MQGET or MQCB using the Hobj returned on the original MQSUB verb that created the subscription.

2. If the topic subscribed to by the original MQSUB verb that created the subscription contained a wildcard, more than one retained publication can be sent. The number of publications sent as a result of this call is recorded in the NumPubs field in the SubRqOpts structure.
3. If this verb completes with a reason code of MQRC_NO_RETAINED_MSG then there were no currently retained publications for the topic specified.#
4. If this verb completes with a reason code of MQRC_RETAINED_MSG_Q_ERROR or MQRC_RETAINED_NOT_DELIVERED then there are currently retained publications for the topic specified but an error has occurred that that meant they were unable to be delivered.
5. The application must have a current subscription to the topic before it can make this call. If the subscription was made in a previous instance of the application and a valid handle to the subscription is not available, the application must first call MQSUB with the MQSO_RESUME option to obtain a handle to it for use in this call.
6. The publications are sent to the destination that is registered for use with the current subscription of this application. If the publications must be sent somewhere else, the subscription must first be altered using the MQSUB call with the MQSO_ALTER option.

C invocation

```
MQSUB (Hconn, Hsub, Action, &SubRqOpts, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHCONN Hconn; /* Connection handle */
MQHOBJ Hsub; /* Subscription handle */
MQLONG Action; /* Action requested by MQSUBRQ */
MQSRO SubRqOpts; /* Options that control the action of MQSUBRQ */
MQLONG CompCode; /* Completion code */
MQLONG Reason; /* Reason code qualifying CompCode */
```

COBOL invocation

```
CALL 'MQSUBRQ' USING HCONN, HSUB, ACTION, SUBRQOPTS, COMPCODE, REASON.
```

Declare the parameters as follows:

```
** Connection handle
01 HCONN PIC S9(9) BINARY.
** Subscription handle
01 HSUB PIC S9(9) BINARY.
** Action requested by MQSUBRQ
01 ACTION PIC S9(9) BINARY.
** Options that control the action of MQSUBRQ
01 SUBRQOPTS.
COPY CMQSROV.
** Completion code
01 COMPCODE PIC S9(9) BINARY.
** Reason code qualifying COMPCODE
01 REASON PIC S9(9) BINARY.
```

PL/I invocation

```
call MQSUBRQ (Hconn, Hsub, Action, SubRqOpts, CompCode, Reason)
```

Declare the parameters as follows:

```
dcl Hconn fixed bin(31); /* Connection handle */
dcl Hsub fixed bin(31); /* Subscription handle */
dcl Action fixed bin(31); /* Action requested by MQSUBRQ */
dcl SubRqOpts like MQSRO; /* Options that control the action of MQSUBRQ */
dcl CompCode fixed bin(31); /* Completion code */
dcl Reason fixed bin(31); /* Reason code qualifying CompCode */
```

High Level Assembler invocation

```
CALL MQSUBRQ,(HCONN, HSUB, ACTION, SUBRQOPTS,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN DS F Connection handle
HSUB DS F Subscription handle
ACTION DS F Action requested by MQSUBRQ
SUBRQOPTS CMQSROA , Options that control the action of MQSUBRQ
COMPCODE DS F Completion code
REASON DS F Reason code qualifying COMPCODE
```

Attributes of objects

This collection of topics lists only those WebSphere MQ objects that can be the subject of an MQINQ function call, and gives details of the attributes that can be inquired on and the selectors to be used.

Attributes for the queue manager:

Some queue-manager attributes are fixed for particular implementations; others can be changed by using the MQSC command `ALTER QMGR`.

The attributes can also be displayed by using the command `DISPLAY QMGR`. Most queue-manager attributes can be inquired by opening a special MQOT_Q_MGR object, and using the MQINQ call with the handle returned.

The following table summarizes the attributes that are specific to the queue manager. The attributes are described in alphabetical order.

Note: The names of the attributes shown in this section are descriptive names used with the MQINQ call; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) Commands for more information.

Table 219. Attributes for the queue manager.

List of queue manager attributes with links and short description

Attribute	Description
AccountingConnOverride	Override accounting settings.
AccountingInterval	How often to write intermediate accounting records.
ActivityConnOverride	Override activity settings.
ActivityTrace	Controls the collection of WebSphere MQ MQI application activity trace.
AdoptNewMCACheck	Elements checked to determine whether to adopt new MCA.
AdoptNewMCAType	Whether to restart automatically an orphaned instance of an MCA of a particular channel type.
AlterationDate	Date when definition was last changed
AlterationTime	Time when definition was last changed
AuthorityEvent	Controls whether authorization (Not Authorized) events are generated
BridgeEvent	Control attribute for bridge events.
ChannelAutoDef	Controls whether automatic channel definition is permitted
ChannelAutoDefEvent	Controls whether channel automatic-definition events are generated
ChannelAutoDefExit	Name of user exit for automatic channel definition
ChannelEvent	Control attribute for channel events.
ChannelInitiatorControl	Control attribute for channel initiator
ChannelMonitoring	Online monitoring data for channels
ChannelStatistics	Controls collection of statistics data for channels.
ChinitAdapters	Number of adapter subtasks for processing WebSphere MQ calls.
ChinitDispatchers	Number of dispatchers to use for the channel initiator.

Table 219. Attributes for the queue manager (continued).

List of queue manager attributes with links and short description

Attribute	Description
	Reserved for IBM use.
ChinitTraceAutoStart	Whether channel initiator trace should start automatically.
ChinitTraceTableSize	Size of channel initiator's trace data space.
ClusterSenderMonitoringDefault	Online monitoring data default for cluster sender channels
ClusterSenderStatistics	Controls collection of statistics monitoring information for cluster sender channels.
ClusterWorkloadData	User data for cluster workload exit
ClusterWorkloadExit	Name of user exit for cluster workload management
ClusterWorkloadLength	Maximum length of message data passed to cluster workload exit
CLWLMRUChannels	Number of most recently used channels for cluster workload balancing
CLWLUseQ	Cluster workload use remote queue.
CodedCharSetId	Coded character set identifier
CommandEvent	Control attribute for command events.
CommandInputQName attribute	Command input queue name
CommandLevel	Command level
CommandServerControl attribute	Control attribute for command server.
Configuration Event attribute	Control attribute for configuration events.
DeadLetterQName	Name of dead-letter queue
DEFCLXQ	Default cluster transmission queue type
DefXmitQName	Default transmission queue name
DistLists	Distribution list support
DNSGroup	Name of group for TCP listener when using Workload Manager Dynamic Domain Name Services support.
DNSWLM	Whether TCP listener registers with Workload Manager for Dynamic Domain Name Services.
ExpiryInterval	Interval between scans for expired messages
IGQPutAuthority	Intra-group queuing put authority
IGQUserId	Intra-group queuing user identifier
InhibitEvent	Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated
IPAddressVersion	Version of the Internet Protocol address
IntraGroupQueuing	Intra-group queuing support
ListenerTimer	Time interval between attempts to restart listener after APPC or TCP/IP failure.
LocalEvent	Controls whether local error events are generated
LoggerEvent	Controls whether logger events are generated
LUGroupName	Generic LU name for LU 6.2 listener that handles inbound transmissions for queue-sharing group.
LUName	Name of LU to use for outbound LU 6.2 transmissions.
LU62ARMSuffix	Suffix of SYS1.PARMLIB member APPCPMxx, that nominates LUADD for this channel initiator.
LU62Channels	Maximum number of current channels or connected clients that use LU 6.2.
MaxActiveChannels	Maximum number of channels that can be active at any time.
MaxChannels	Maximum number of current channels.
MaxHandles	Maximum number of handles
MaxMsgLength	Maximum message length in bytes
MaxPriority attribute	Maximum priority
MaxPropertiesLength	Maximum length of property data in bytes
MaxUncommittedMsgs	Maximum number of uncommitted messages within a unit of work
MQIAccounting	Controls collection of accounting information for MQI data.
MQIStatistics	Controls collection of statistics monitoring information for queue manager.
MsgMarkBrowseInterval	Interval after which the queue manager can remove the mark from browsed messages.
OutboundPortMin	With *OutboundPortMin*, defines range of port numbers to use when binding outgoing channels.
OutboundPortMin	With *OutboundPortMax*, defines range of port numbers to use when binding outgoing channels.
PerformanceEvent	Controls whether performance-related events are generated

Table 219. Attributes for the queue manager (continued).

List of queue manager attributes with links and short description

Attribute	Description
Platform	Platform on which the queue manager is running
PubSubNPInputMsg	Whether to discard (or keep) an undelivered input message
PubSubNPResponse	Controls the behavior of undelivered
PubSubMaxMsgRetryCount	The number of retries when processing (under syncpoint) a failed command message
PubSubSyncPoint	Whether only persistent (or all) messages should be processed under syncpoint
PubSubMode	Whether the queued publish/subscribe interface is running
QMgrDesc	Queue manager description
QMgrIdentifier	Unique internally generated identifier of queue manager
QMgrName	Queue manager name
QSGName	Name of queue-sharing group
QueueAccounting	Controls collection of accounting information for queues.
QueueMonitoring	Online monitoring data for queues
QueueStatistics	Controls collection of statistics data for queues.
ReceiveTimeout	How long TCP/IP channel waits for data before returning to inactive state.
ReceiveTimeoutMin	Qualifier for *ReceiveTimeout*.
ReceiveTimeoutType	Minimum time that TCP/IP channel waits for data before returning to inactive state.
RemoteEvent	Controls whether remote error events are generated
RepositoryName	Name of cluster for which this queue manager provides repository services
RepositoryNamelist	Name of namelist object containing names of clusters for which this queue manager provides repository services
ScyCase	Case of security profiles
SharedQMgrName	Shared queue queue-manager name
"SPLCAP" on page 2130	WebSphere MQ Advanced Message security protection for a queue manager turned on or off.
SSLCRLNamelist [1]	Name of namelist object containing names of authentication information objects.
SSLCryptoHardware [1]	Cryptographic hardware configuration string.
SSLEvent	Control attribute for SSL events.
SSLFIPSRequired	Use only FIPS-certified algorithms for cryptography.
SSLKeyRepository [1]	Location of SSL key repository.
SSLKeyResetCount	SSL key reset count.
SSLTasks [1]	Number of server subtasks for processing SSL calls.
StatisticsInterval	How often to write statistics monitoring data.
StartStopEvent	Controls whether start and stop events are generated
SyncPoint	Syncpoint availability
TCPChannels	Maximum number of current channels or connected clients that use TCP/IP.
TCPKeepAlive	Whether to use TCP KEEPALIVE to check other end of connection.
TCPName	Name of TCP/IP system that you are using.
TCPStackType	How channel initiator can use TCP/IP addresses.
TraceRouteRecording attribute	Controls recording of trace-route information.
TriggerInterval	Trigger-message interval
Version	Version
XrCapability	Specifies whether Telemetry commands are supported.

Notes:

1. This attribute cannot be inquired using the MQINQ call, and is not described in this section. See Change Queue Manager for details of this attribute.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

AccountingConnOverride (MQLONG):

This allows applications to override the setting of the ACCTMQI and ACCTQDATA values in the Qmgr attribute.

The value is one of the following:

MQMON_DISABLED
: Applications cannot override the setting of the ACCTMQI and ACCTQ Qmgr attributes using the Options field in the MQCNO structure on the MQCONNX call. This is the default value.

MQMON_ENABLED
: Applications can override the ACCTQ and ACCTMQI Qmgr attributes using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This attribute is supported only on IBM i, Unix systems, and Windows.

To determine the value of this attribute, use the MQIA_ACCOUNTING_CONN_OVERRIDE selector with the MQINQ call.

AccountingInterval (MQLONG):

This specifies how long before intermediate accounting records are written (in seconds).

The value is an integer in the range 0 to 604800, with a default value of 1800 (30 minutes). Specify 0 to turn off intermediate records.

This attribute is supported only on IBM i, Windows, UNIX, and Linux systems.

To determine the value of this attribute, use the MQIA_ACCOUNTING_INTERVAL selector with the MQINQ call.

ActivityConnOverride (MQLONG):

This allows applications to override the setting of the ACTVTRC value in the queue manager attribute.

The value is one of the following:

MQMON_DISABLED
: Applications cannot override the setting of the `ACTVTRC` queue manager attribute using the Options field in the MQCNO structure on the MQCONNX call. This is the default value.

MQMON_ENABLED
: Applications can override the `ACTVTRC` queue manager attribute using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This attribute is supported only on IBM i, Unix systems, and Windows.

To determine the value of this attribute, use the MQIA_ACTIVITY_CONN_OVERRIDE selector with the MQINQ call.

ActivityTrace (MQLONG):

This controls the collection of WebSphere MQ MQI application activity trace.

The value is one of the following:

MQMON_ON
Collect WebSphere MQ MQI application activity trace.

MQMON_OFF
Do not collect WebSphere MQ MQI application activity trace. This is the default value.

If you set the queue manager attribute ACTVCONO to ENABLED, this value might be overridden for individual connections using the Options field in the MQCNO structure.

Changes to this value are only effective for connections to the queue manager after the change to the attribute.

This attribute is supported only on IBM i, Unix systems, and Windows.

To determine the value of this attribute, use the MQIA_ACTIVITY_TRACE selector with the MQINQ call.

AdoptNewMCACheck (MQLONG):

This defines the elements to check to determine whether to adopt an MCA when a new inbound channel is detected that has the same name as an MCA that is already active

The value is one of the following:

MQADOPT_CHECK_Q_MGR_NAME
Check the queue manager name.

MQADOPT_CHECK_NET_ADDR
Check the network address.

MQADOPT_CHECK_ALL
Check the queue manager name and network address. If possible, perform this check to protect your channels from being shut down, inadvertently or maliciously. This is the default value.

MQADOPT_CHECK_NONE
Do not check any elements.

Changes to this attribute take effect the next time that a channel attempts to adopt a channel.

This attribute is supported only on z/OS.

To determine the value of this attribute, use the MQIA_ADOPTNEWMCA_CHECK selector with the MQINQ call.

AdoptNewMCAType (MQLONG):

This specifies whether to restart automatically an orphaned instance of an MCA of a particular channel type when a new inbound channel request matching the AdoptNewMCACheck attribute is detected

It is one of the following values:

MQADOPT_TYPE_NO
Adopting orphaned channel instances is not required. This is the default value.

MQADOPT_TYPE_ALL
Adopt all channel types.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_ADOPTNEWMCA_TYPE selector with the MQINQ call.

AlterationDate (MQCHAR12):

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8):

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

AuthorityEvent (MQLONG):

This controls whether authorization (Not Authorized) events are generated. It is one of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_AUTHORITY_EVENT selector with the MQINQ call.

BridgeEvent (MQLONG):

This specifies whether IMS bridge events are generated.

The value is one of the following:

MQEVR_ENABLED
Generate IMS bridge events, as follows:
MQRC_BRIDGE_STARTED
MQRC_BRIDGE_STOPPED

MQEVR_DISABLED
Do not generate IMS bridge events; this is the default value.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_BRIDGE_EVENT selector with the MQINQ call.

ChannelAutoDef (MQLONG):

This attribute controls the automatic definition of channels of type MQCHT_RECEIVER and MQCHT_SVRCONN. Automatic definition of MQCHT_CLUSSDR channels is always enabled. The value is one of the following:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

To determine the value of this attribute, use the MQIA_CHANNEL_AUTO_DEF selector with the MQINQ call.

ChannelAutoDefEvent (MQLONG):

This controls whether channel automatic-definition events are generated. It applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

To determine the value of this attribute, use the MQIA_CHANNEL_AUTO_DEF_EVENT selector with the MQINQ call.

ChannelAutoDefExit (MQCHARn):

This is the name of the user exit for automatic channel definition. If this name is nonblank, and *ChannelAutoDef* has the value MQCHAD_ENABLED, the exit is called each time that the queue manager is about to create a channel definition. This applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The exit can then do one of the following:

- Create the channel definition without change.
- Modify the attributes of the channel definition that is created.
- Suppress creation of the channel entirely.

Note: Both the length and the value of this attribute are environment specific. See the introduction to the MQCD structure in "MQCD - Channel definition" on page 2348 for details of the value of this attribute in various environments.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS. On z/OS, it applies only to cluster-sender and cluster-receiver channels.

To determine the value of this attribute, use the MQCA_CHANNEL_AUTO_DEF_EXIT selector with the MQINQ call. The length of this attribute is given by MQ_EXIT_NAME_LENGTH.

ChannelEvent (MQLONG):

This specifies whether channel events are generated.

It is one of the following values:

MQEVR_EXCEPTION
Only generate the following channel events:
- MQRC_CHANNEL_ACTIVATED
- MQRC_CHANNEL_CONV_ERROR
- MQRC_CHANNEL_NOT_ACTIVATED
- MQRC_CHANNEL_STOPPED with the following ReasonQualifiers:
  MQRQ_CHANNEL_STOPPED_ERROR
  MQRQ_CHANNEL_STOPPED_RETRY
  MQRQ_CHANNEL_STOPPED_DISABLED

  MQRC_CHANNEL_STOPPED_BY_USER

MQEVR_ENABLED
Generate all channel events. That is, in addition to those generated by EXCEPTION, generate the following channel events:
- MQRC_CHANNEL_STARTED
- MQRC_CHANNEL_STOPPED with the following ReasonQualifier:
  MQRQ_CHANNEL_STOPPED_OK

MQEVR_DISABLED
Do not generate channel events; this is the default value.

To determine the value of this attribute, use the MQIA_CHANNEL_EVENT selector with the MQINQ call.

ChannelInitiatorControl (MQLONG):

This specifies whether the channel initiator is to be started when the queue manager starts.

It is one of the following values:

MQSVC_CONTROL_MANUAL
The channel initiator is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The channel initiator is to be started automatically when the queue manager starts.

To determine the value of this attribute, use the MQIA_CHINIT_CONTROL selector with the MQINQ call.

ChannelMonitoring (MQLONG):

This specifies online monitoring data for channels.

The value is one of the following:

MQMON_NONE
Disable data collection for channel monitoring for all channels regardless of the setting of the MONCHL channel attribute. This is the default value.

MQMON_OFF
Turn monitoring data collection off for channels that specify QMGR in the MONCHL channel attribute.

MQMON_LOW
Turn monitoring data collection on with a low ratio of data collection for channels specifying QMGR in the MONCHL channel attribute.

MQMON_MEDIUM
Turn monitoring data collection on with a moderate ratio of data collection for channels specifying QMGR in the MONCHL channel attribute.

MQMON_HIGH
Turn monitoring data collection on with a high ratio of data collection for channels specifying QMGR in the MONCHL channel attribute.

To determine the value of this attribute, use the MQIA_MONITORING_CHANNEL selector with the MQINQ call.

ChannelStatistics (MQLONG):

This controls the collection of statistics data for channels.

The value is one of the following:

MQMON_NONE
Disable data collection for channel statistics for all channels regardless of the setting of the STATCHL channel attribute. This is the default value.

MQMON_OFF
Turn statistics data collection off for channels that specify QMGR in the STATCHL channel attribute.

MQMON_LOW
Turn statistics data collection on with a low ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

MQMON_MEDIUM
Turn statistics data collection on with a moderate ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

MQMON_HIGH
Turn statistics data collection on with a high ratio of data collection for channels specifying QMGR in the STATCHL channel attribute.

For most systems you are recommended to use MEDIUM. However, for a channel that processes a high volume of messages each second, you might want to reduce the sampling level by selecting LOW. Also, for a channel that processes only a few messages, and for which the most current information is important, you might want to select HIGH.

This attribute is supported only on IBM i, UNIX systems, and Windows.

To determine the value of this attribute, use the MQIA_STATISTICS_CHANNEL selector with the MQINQ call.

ChinitAdapters (MQLONG):

This is the number of adapter subtasks to use to process WebSphere MQ calls. The value must be 0 - 9999, with a default value of 8.

The ratio of adapters to dispatchers (the ChinitDispatchers attribute) should be about 8 to 5. However, if you have only few channels, you do not have to decrease the value of this parameter from the default value. You can use the following values: for a test system, 8 (default); for a production system, 20. Ideally,

you should have 20 adapters, which gives greater parallelism of WebSphere MQ calls. This is important for persistent messages. Fewer adapters might be better for nonpersistent messages.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_ADAPTERS selector with the MQINQ call.

ChinitDispatchers (MQLONG):

This is the number of dispatchers to use for the channel initiator. The value must be 0 - 9999, with a default value of 5.

As a guideline, allow one dispatcher for 50 current channels. However, if you have only few channels, you do not have to decrease the value of this attribute from the default value. If you are using TCP/IP, the greatest number of dispatchers that are used for TCP/IP channels is 100, even if you specify a larger value here. You can use the following settings: test systems, 5 (the default); production systems, 20 (you need 20 dispatchers to handle up to 1000 active channels).

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_DISPATCHERS selector with the MQINQ call.

ChinitTraceAutoStart (MQLONG):

This specifies whether to start channel initiator trace automatically.

The value is one of the following:

MQTRAXSTR_YES
Start channel initiator trace automatically. This is the default value.

MQTRAXSTR_NO
Do not start channel initiator trace automatically.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_TRACE_AUTO_START selector with the MQINQ call.

ChinitTraceTableSize (MQLONG):

This is the size of the channel initiator's trace data space (in MB).

The value must be in the range 0 through 2048, with a default value of 2.

Note: Whenever you use large z/OS data spaces, ensure that you have sufficient auxiliary storage on your system to support any related z/OS paging activity. You might also need to increase the size of your SYS1.DUMP data sets.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_CHINIT_TRACE_TABLE_SIZE selector with the MQINQ call.

ClusterSenderMonitoringDefault (MQLONG):

This specifies the value to be substituted for the ChannelMonitoring attribute of automatically-defined cluster sender channels.

The value is one of the following:

MQMON_Q_MGR
Collection of online monitoring data is inherited from the setting of the queue manager *ChannelMonitoring* attribute. This is the default value.

MQMON_OFF
Monitoring for the channel is switched off

MQMON_LOW
Unless *ChannelMonitoring* is MQMON_NONE, monitoring is switched on with a low rate of data collection with a minimal effect on system performance. The data collected is not likely to be the most current.

MQMON_MEDIUM
Unless *ChannelMonitoring* is MQMON_NONE, monitoring is switched on with a moderate rate of data collection with limited effect on system performance.

MQMON_HIGH
Unless *ChannelMonitoring* is MQMON_NONE, monitoring is switched on with a high rate of data collection with a likely effect on system performance. The data collected is the most current available.

To determine the value of this attribute, use the MQIA_MONITORING_AUTO_CLUSSDR selector with the MQINQ call.

ClusterSenderStatistics (MQLONG):

Because cluster sender channels can be automatically defined from the definition of CLUSRCVR in the repository, you cannot alter the setting of the STATCHL attribute for these auto-defined cluster sender channels using ALTER channel. For these channels the decision of whether to collect online monitoring data is based on the setting of this queue manager attribute.

The value is one of the following:

MQMON_Q_MGR
Statistics data collection for auto-defined cluster sender channels is based on the value of the queue manager attribute STATCHL. This is the default value.

MQMON_OFF
Switch off statistics data collection for auto-defined cluster sender channels.

MQMON_LOW
Switch on statistics data collection for auto-defined cluster sender channels with a low ratio of data collection.

MQMON_MEDIUM
Switch on statistics data collection for auto-defined cluster sender channels with a moderate ratio of data collection.

MQMON_HIGH
Switch on statistics data collection for auto-defined cluster sender channels with a high ratio of data collection.

For most systems we recommend MEDIUM. However, for an auto-defined cluster sender channel that processes a high volume of messages each second, you might want to reduce the sampling level by

selecting LOW. Also, for a channel that processes only a few messages, and for which the most current information is important, you might want to select HIGH.

To determine the value of this attribute, use the MQIA_STATISTICS_AUTO_CLUSSDR selector with the MQINQ call.

ClusterWorkloadData (MQCHAR32):

This is a user-defined 32-byte character string that is passed to the cluster workload exit when it is called. If there is no data to pass to the exit, the string is blank.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, Windows and z/OS.

To determine the value of this attribute, use the MQCA_CLUSTER_WORKLOAD_DATA selector with the MQINQ call.

ClusterWorkloadExit (MQCHARn):

This is the name of the user exit for cluster workload management. If this name is not blank, the exit is called each time that a message is put to a cluster queue or moved from one cluster-sender queue to another. The exit can then either accept the queue instance selected by the queue manager as the destination for the message, or select another queue instance.

Note: Both the length and the value of this attribute are environment specific.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, Windows and z/OS.

To determine the value of this attribute, use the MQCA_CLUSTER_WORKLOAD_EXIT selector with the MQINQ call. The length of this attribute is given by MQ_EXIT_NAME_LENGTH.

ClusterWorkloadLength (MQLONG):

This is the maximum length of message data that is passed to the cluster workload exit. The actual length of data passed to the exit is the minimum of the following:

- The length of the message.
- The queue-manager's *MaxMsgLength* attribute.
- The *ClusterWorkloadLength* attribute.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, Windows and z/OS.

To determine the value of this attribute, use the MQIA_CLUSTER_WORKLOAD_LENGTH selector with the MQINQ call.

CLWLMRUChannels (MQLONG):

This specifies the maximum number of most-recently-used cluster channels, to be considered for use by the cluster workload choice algorithm.

This is a value in the range 1 through 999999999.

To determine the value of this attribute, use the MQIA_CLWL_MRU_CHANNELS selector with the MQINQ call.

CLWLUseQ (MQLONG):

This specifies whether to use remote queues for the cluster workload.

The value is one of the following:

MQCLWL_USEQ_ANY
Use both local and remote queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues. This is the default value.

To determine the value of this attribute, use the MQIA_CLWL_USEQ selector with the MQINQ call.

CodedCharSetId (MQLONG):

This defines the character set used by the queue manager for all character string fields defined in the MQI such as the names of objects, and queue creation date and time. The character set must be one that has single-byte characters for the characters that are valid in object names. It does not apply to application data carried in the message. The value depends on the environment:

- On z/OS, the value is set from the system parameters when the queue manager is started; the default value is 500.
- On Windows, the value is the primary CODEPAGE of the user creating the queue manager.
- On IBM i, the value is that which is set in the environment when the queue manager is first created.
- On UNIX systems, the value is the default CODESET for the locale of the user creating the queue manager.

To determine the value of this attribute, use the MQIA_CODED_CHAR_SET_ID selector with the MQINQ call.

CommandEvent (MQLONG):

This specifies whether command events are generated, as follows:

MQEVR_DISABLED
Do not generate command events. This is the default.

MQEVR_ENABLED
Generate command events.

MQEVR_NO_DISPLAY
Command events are generated for all successful commands other than MQINQ.

To determine the value of this attribute, use the MQIA_COMMAND_EVENT selector with the MQINQ call.

CommandInputQName (MQCHAR48):

This is the name of the command input queue defined on the local queue manager. This is a queue to which users can send commands, if authorized to do so. The name of the queue depends on the environment:

- On z/OS, the name of the queue is SYSTEM.COMMAND.INPUT; MQSC and PCF commands can be sent to it. See The MQSC commands for details of MQSC commands and Definitions of the Programmable Command Formats for details of PCF commands.

- In all other environments, the name of the queue is SYSTEM.ADMIN.COMMAND.QUEUE, and only PCF commands can be sent to it. However, an MQSC command can be sent to this queue if the MQSC command is enclosed within a PCF command of type MQCMD_ESCAPE. See Escape for information about the Escape command.

To determine the value of this attribute, use the MQCA_COMMAND_INPUT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

CommandLevel (MQLONG):

This indicates the level of system control commands supported by the queue manager. This can be one of the following values:

MQCMDL_LEVEL_1
Level 1 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- MQSeries for AIX Version 2 Release 2
- MQSeries for
  - Version 1 Release 1.1
  - Version 1 Release 1.2
  - Version 1 Release 1.3
- MQSeries for OS/400
  - Version 2 Release 3
  - Version 3 Release 1
  - Version 3 Release 6
- MQSeries for Windows Version 2 Release 0

MQCMDL_LEVEL_101
MQSeries for Windows Version 2 Release 0.1.

MQCMDL_LEVEL_110
MQSeries for Windows Version 2 Release 1.

MQCMDL_LEVEL_114
MQSeries for Version 1 Release 1.4.

MQCMDL_LEVEL_120
MQSeries for Version 1 Release 2.0.

MQCMDL_LEVEL_200
MQSeries for Windows NT Version 2 Release 0.

MQCMDL_LEVEL_210
MQSeries for OS/390 Version 2 Release 1.0.

MQCMDL_LEVEL_220
Level 220 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- MQSeries for AT&T GIS UNIX Version 2 Release 2
- MQSeries for SINIX and DC/OSx Version 2 Release 2
- MQSeries for SunOS Version 2 Release 2
- MQSeries for Tandem NonStop Kernel Version 2 Release 2

MQCMDL_LEVEL_221
Level 221 of system control commands.

This value is returned by MQSeries for AIX Version 2 Release 2.1

MQCMDL_LEVEL_320
Level 320 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- MQSeries for OS/400
  - Version 3 Release 2
  - Version 3 Release 7

MQCMDL_LEVEL_420
Level 420 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- MQSeries for IBM i
  - Version 4 Release 2.0
  - Version 4 Release 2.1

MQCMDL_LEVEL_500
Level 500 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- IBM WebSphere MQ for AIX Version 5 Release 0
- MQSeries for HP-UX Version 5 Release 0
- MQSeries for Solaris Version 5 Release 0
- MQSeries for Windows NT Version 5 Release 0

MQCMDL_LEVEL_510
Level 510 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- IBM WebSphere MQ for AIX Version 5 Release 1
- MQSeries for AS/400 Version 5 Release 1
- MQSeries for HP-UX Version 5 Release 1
- IBM WebSphere MQ for HP Integrity NonStop Server Version 5 Release 3
- MQSeries for Compaq Tru64 UNIX Version 5 Release 1
- MQSeries for Solaris Version 5 Release 1
- MQSeries for Windows NT Version 5 Release 1

MQCMDL_LEVEL_520
Level 520 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- MQSeries for AIX ersion 5 Release 2
- MQSeries for AS/400 Version 5 Release 2
- MQSeries for HP-UX Version 5 Release 2
- MQSeries for Linux Version 5 Release 2
- MQSeries for OS/390 Version 5 Release 2
- MQSeries for Sun Solaris Version 5 Release 2
- MQSeries for Windows NT Version 5 Release 2

MQCMDL_LEVEL_530
Level 530 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- IBM WebSphere MQ for AIX Version 5 Release 3
- IBM WebSphere MQ for HP-UX Version 5 Release 3

- IBM WebSphere MQ for i/Series Version 5 Release 3
- IBM WebSphere MQ for Linux for Intel Version 5 Release 3
- IBM WebSphere MQ for Linux for zSeries Version 5 Release 3
- IBM WebSphere MQ for Solaris Version 5 Release 3
- IBM WebSphere MQ for Windows Version 5 Release 3
- IBM WebSphere MQ for z/OS Version 5 Release 3

MQCMDL_LEVEL_600
Level 600 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:
- IBM WebSphere MQ for AIX Version 6.0
- IBM WebSphere MQ for HP-UX Version 6.0
- IBM WebSphere MQ for i/Series Version 6.0
- IBM WebSphere MQ for Linux Version 6.0
- IBM WebSphere MQ for Solaris Version 6.0
- IBM WebSphere MQ for Windows Version 6.0
- IBM WebSphere MQ for z/OS Version 6.0

MQCMDL_LEVEL_700
Level 700 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:
- IBM WebSphere MQ for AIX Version 7.0
- IBM WebSphere MQ for HP-UX Version 7.0
- IBM WebSphere MQ for IBM i Version 7.0
- IBM WebSphere MQ for Linux Version 7.0
- IBM WebSphere MQ for Solaris Version 7.0
- IBM WebSphere MQ for Windows Version 7.0
- IBM WebSphere MQ for z/OS Version 7.0

MQCMDL_LEVEL_701
Level 701 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:
- IBM WebSphere MQ for AIX Version 7.0.1
- IBM WebSphere MQ for HP-UX Version 7.0.1
- IBM WebSphere MQ for IBM i Version 7.0.1
- IBM WebSphere MQ for Linux Version 7.0.1
- IBM WebSphere MQ for Solaris Version 7.0.1
- IBM WebSphere MQ for Windows Version 7.0.1
- IBM WebSphere MQ for z/OS Version 7.0.1

MQCMDL_LEVEL_710
Level 710 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:
- IBM WebSphere MQ for AIX Version 7.1
- IBM WebSphere MQ for HP-UX Version 7.1
- IBM WebSphere MQ for IBM i Version 7.1
- IBM WebSphere MQ for Linux Version 7.1
- IBM WebSphere MQ for Solaris Version 7.1

- IBM WebSphere MQ for Windows Version 7.1
- IBM WebSphere MQ for z/OS Version 7.1

MQCMDL_LEVEL_750
Level 750 of system control commands.

This value is returned by the following versions of IBM WebSphere MQ:

- IBM WebSphere MQ for AIX Version 7.5
- IBM WebSphere MQ for HP-UX Version 7.5
- IBM WebSphere MQ for IBM i Version 7.5
- IBM WebSphere MQ for Linux Version 7.5
- IBM WebSphere MQ for Solaris Version 7.5
- IBM WebSphere MQ for Windows Version 7.5

The set of system control commands that corresponds to a particular value of the *CommandLevel* attribute varies according to the value of the *Platform* attribute; both must be used to decide which system control commands are supported.

To determine the value of this attribute, use the MQIA_COMMAND_LEVEL selector with the MQINQ call.

CommandServerControl (MQLONG):

Specifies whether the command server is to be started when the queue manager starts.

The value can be:

MQSVC_CONTROL_MANUAL
The command server is not to be started automatically.

MQSVC_CONTROL_Q_MGR
The command server is to be started automatically when the queue manager starts.

This attribute is not supported on z/OS.

To determine the value of this attribute, use the MQIA_CMD_SERVER_CONTROL selector with the MQINQ call.

ConfigurationEvent (MQLONG):

Controls whether configuration events are generated.

To determine the value of this attribute, use the MQIA_CONFIGURATION_EVENT selector with the MQINQ call.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

DeadLetterQName (MQCHAR48):

This is the name of a queue defined on the local queue manager as the dead-letter (undelivered-message) queue. Messages are sent to this queue if they cannot be routed to their correct destination.

For example, messages are put on this queue when:

- A message arrives at a queue manager, destined for a queue that is not yet defined on that queue manager
- A message arrives at a queue manager, but the queue for which it is destined cannot receive it because, possibly:
  - The queue is full
  - Put requests are inhibited
  - The sending node does not have authority to put messages on the queue

Applications can also put messages on the dead-letter queue.

Report messages are treated in the same way as ordinary messages; if the report message cannot be delivered to its destination queue (usually the queue specified by the *ReplyToQ* field in the message descriptor of the original message), the report message is placed on the dead-letter (undelivered-message) queue.

Note: Messages that have passed their expiry time (see MQMD - Expiry field) are **not** transferred to this queue when they are discarded. However, an expiration report message (MQRO_EXPIRATION) is still generated and sent to the *ReplyToQ* queue, if requested by the sending application.

Messages are not put on the dead-letter (undelivered-message) queue when the application that issued the put request has been notified synchronously of the problem by means of the reason code returned by the MQPUT or MQPUT1 call (for example, a message put on a local queue for which put requests are inhibited).

Messages on the dead-letter (undelivered-message) queue sometimes have their application message data prefixed with an MQDLH structure. This structure contains extra information that indicates why the message was placed on the dead-letter (undelivered-message) queue. See “MQDLH - Dead-letter header” on page 1636 for more details of this structure.

This queue must be a local queue, with a *Usage* attribute of MQUS_NORMAL.

If a queue manager does not support a dead-letter (undelivered-message) queue, or one has not been defined, the name is all blanks. All WebSphere MQ queue managers support a dead-letter (undelivered-message) queue, but by default it is not defined.

If the dead-letter (undelivered-message) queue is not defined, full, or unusable for some other reason, a message which would have been transferred to it by a message channel agent is retained instead on the transmission queue.

To determine the value of this attribute, use the MQCA_DEAD_LETTER_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

DefClusterXmitQueueType (MQLONG):

The `DefClusterXmitQueueType` attribute controls which transmission queue is selected by default by cluster-sender channels to get messages from, to send the messages to cluster-receiver channels.

The values of `DefClusterXmitQueueType` are `MQCLXQ_SCTQ` or `MQCLXQ_CHANNEL`.

`MQCLXQ_SCTQ`

All cluster-sender channels send messages from `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. The `correlID` of messages placed on the transmission queue identifies which cluster-sender channel the message is destined for.

SCTQ is set when a queue manager is defined. This behavior is implicit in versions of IBM WebSphere MQ, earlier than Version 7.5. In earlier versions, the queue manager attribute DefClusterXmitQueueType was not present.

MQCLXQ_CHANNEL
Each cluster-sender channel sends messages from a different transmission queue. Each transmission queue is created as a permanent dynamic queue from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE.

The attribute is not supported on z/OS.

If the queue manager attribute, DefClusterXmitQueueType, is set to CHANNEL, the default configuration is changed to cluster-sender channels being associated with individual cluster transmission queues. The transmission queues are permanent-dynamic queues created from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Each transmission queue is associated with one cluster-sender channel. As one cluster-sender channel services a cluster transmission queue, the transmission queue contains messages for only one queue manager in one cluster. You can configure clusters so that each queue manager in a cluster contains only one cluster queue. In this case, the message traffic from a queue manager to each cluster queue is transferred separately from messages to other queues.

To query the value, call MQINQ, or send an Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) PCF command, setting the MQIA_DEF_CLUSTER_XMIT_Q_TYPE selector. To change the value, send a Change Queue Manager (MQCMD_CHANGE_Q_MGR) PCF command, setting the MQIA_DEF_CLUSTER_XMIT_Q_TYPE selector.

Related reference:

"MQINQ - Inquire object attributes" on page 2005
The MQINQ call returns an array of integers and a set of character strings containing the attributes of an object.

Related information:

Change Queue Manager
The Change Queue Manager (MQCMD_CHANGE_Q_MGR) command changes the specified attributes of the queue manager.

Inquire Queue Manager
The Inquire Queue Manager (**MQCMD_INQUIRE_Q_MGR**) command inquires about the attributes of a queue manager.

DefXmitQName (MQCHAR48):

This is the name of the transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

If there is no default transmission queue, the name is entirely blank. The initial value of this attribute is blank.

To determine the value of this attribute, use the MQCA_DEF_XMIT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

DistLists (MQLONG):

This indicates whether the local queue manager supports distribution lists on the MQPUT and MQPUT1 calls. It is one of the following values:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call.

DNSGroup (MQCHAR18):

This is the name of the group for the TCP listener that handles inbound transmissions for the queue-sharing group to join when using Workload Manager Dynamic Domain Name Services support. The maximum length is 18 characters. If you leave this name blank, the queue-sharing group name is used.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_DNS_GROUP selector with the MQINQ call. The length of this attribute is given by MQ_DNS_GROUP_NAME_LENGTH.

DNSWLM (MQLONG):

This specifies whether the TCP listener that handles inbound transmissions for the queue-sharing group registers with Workload Manager for Dynamic Domain Name Services

The value is one of the following:

MQDNSWLM_YES
The listener registers with Workload Manager.

MQDNSWLM_NO
The listener does not register with Workload Manager. This is the default value.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_DNS_WLM selector with the MQINQ call.

ExpiryInterval (MQLONG):

This indicates the frequency with which the queue manager scans the queues looking for expired messages. It is either a time interval in seconds in the range 1 through 99 999 999, or the following special value:

MQEXPI_OFF
The queue manager does not scan the queues looking for expired messages.

To determine the value of this attribute, use the MQIA_EXPIRY_INTERVAL selector with the MQINQ call.

This attribute is supported only on z/OS.

IGQPutAuthority (MQLONG):

This attribute applies only if the local queue manager is a member of a queue-sharing group. It indicates the type of authority checking that is performed when the local intra-group queuing agent (IGQ agent) removes a message from the shared transmission queue and places the message on a local queue. The value is one of the following:

MQIGQPA_DEFAULT
The user identifier checked for authorization is the value of the *UserIdentifier* field in the *separate* MQMD that is associated with the message when the message is on the shared transmission queue. This is the user identifier of the program that placed the message on the shared transmission queue, and is usually the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) is also checked.

MQIGQPA_CONTEXT
The user identifier checked for authorization is the value of the *UserIdentifier* field in the *separate* MQMD that is associated with the message when the message is on the shared transmission queue. This is the user identifier of the program that placed the message on the shared transmission queue, and is usually the same as the user identifier under which the remote queue manager is running.

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the user identifier of the local IGQ agent (*IGQUserId*) and the value of the *UserIdentifier* field in the *embedded* MQMD are also checked. The latter user identifier is usually the user identifier of the application that originated the message.

MQIGQPA_ONLY_IGQ
The user identifier checked for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, this user identifier is used for all checks.

MQIGQPA_ALTERNATE_OR_IGQ
The user identifier checked for authorization is the user identifier of the local IGQ agent (*IGQUserId*).

If the RESLEVEL profile indicates that more than one user identifier is to be checked, the value of the *UserIdentifier* field in the *embedded* MQMD is also checked. This user identifier is usually the user identifier of the application that originated the message.

To determine the value of this attribute, use the MQIA_IGQ_PUT_AUTHORITY selector with the MQINQ call.

This attribute is supported only on z/OS.

IGQUserId (MQLONG):

This attribute is applicable only if the local queue manager is a member of a queue-sharing group. It specifies the user identifier that is associated with the local intra-group queuing agent (IGQ agent). This identifier is one of the user identifiers that can be checked for authorization when the IGQ agent puts messages on local queues. The actual user identifiers checked depend on the setting of the *IGQPutAuthority* attribute, and on external security options.

If *IGQUserId* is blank, no user identifier is associated with the IGQ agent and the corresponding authorization check is not performed (although other user identifiers might still be checked for authorization).

To determine the value of this attribute, use the MQCA_IGQ_USER_ID selector with the MQINQ call. The length of this attribute is given by MQ_USER_ID_LENGTH.

This attribute is supported only on z/OS.

InhibitEvent (MQLONG):

This controls whether inhibit (Inhibit Get and Inhibit Put) events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_INHIBIT_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

IntraGroupQueuing (MQLONG):

This attribute applies only if the local queue manager is a member of a queue-sharing group. It indicates whether intra-group queuing is enabled for the queue-sharing group. The value is one of the following:

MQIGQ_DISABLED
All messages destined for other queue managers in the queue-sharing group are transmitted using conventional channels..

MQIGQ_ENABLED
Messages destined for other queue managers in the queue-sharing group are transmitted using the shared transmission queue if the following condition is satisfied:

- The length of the message data plus transmission header does not exceed 63 KB (64 512 bytes).

  It is recommended that somewhat more space than the size of MQXQH be allocated for the transmission header; the constant MQ_MSG_HEADER_LENGTH is provided for this purpose.

If this condition is not satisfied, the message is transmitted using conventional channels.

Note: When intra-group queuing is enabled, the order of messages transmitted using the shared transmission queue is not preserved relative to those transmitted using conventional channels.

To determine the value of this attribute, use the MQIA_INTRA_GROUP_QUEUING selector with the MQINQ call.

This attribute is supported only on z/OS.

IPAddressVersion (MQLONG):

Specifies which IP address version, either IPv4 or IPv6, is used.

This attribute is only relevant for systems that run both IPv4 and IPv6 and only affects channels defined as having a *TransportType* of MQXPY_TCP when one of the following conditions is true:

- The channel's *ConnectionName* is a host name that resolves to both an IPv4 and IPv6 address and its *LocalAddress* parameter is not specified.
- The channel's *ConnectionName* and *LocalAddress* are both host names that resolve to both IPv4 and IPv6 addresses.

The value can be:

MQIPADDR_IPV4
IPv4 is used.

MQIPADDR_IPV6
IPv6 is used.

To determine the value of this attribute, use the MQIA_IP_ADDRESS_VERSION selector with the MQINQ call.

ListenerTimer (MQLONG):

This is the time interval (in seconds) between WebSphere MQ attempts to restart the listener if there has been an APPC or TCP/IP failure. The value must be between 5 and 9999, with a default value of 60.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_LISTENER_TIMER selector with the MQINQ call.

LocalEvent (MQLONG):

This controls whether local error events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_LOCAL_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

LoggerEvent (MQLONG):

This controls whether recovery log events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_LOGGER_EVENT selector with the MQINQ call.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, and Windows.

LUGroupName (MQCHAR8):

This is the generic LU name for the LU 6.2 listener that handles inbound transmissions for the queue-sharing group. If you leave this name blank, you cannot use this listener.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU_GROUP_NAME selector with the MQINQ call. The length of this attribute is given by MQ_LU_NAME_LENGTH.

LUName (MQCHAR8):

This is the name of the LU to use for outbound LU 6.2 transmissions. Set this to the same LU that the listener uses for inbound transmissions. If you leave this name blank, the APPC/MVS default LU is used; this is variable, so always set LUName if you are using LU6.2.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU_NAME selector with the MQINQ call. The length of this attribute is given by MQ_LU_NAME_LENGTH.

LU62ARMSuffix (MQCHAR2):

This is the suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator. The z/OS command SET APPC=xx is issued when ARM restarts the channel initiator. If you leave this name is blank, no SET APPC=xx is issued.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_LU62_ARM_SUFFIX selector with the MQINQ call. The length of this attribute is given by MQ_ARM_SUFFIX_LENGTH.

LU62Channels (MQLONG):

This is the maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

The value must be in the range 0 through 9999, with a default value of 200. If you set this to zero, the LU 6.2 transmission protocol is not used.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_LU62_CHANNELS selector with the MQINQ call.

MaxActiveChannels (MQLONG):

This attribute is the maximum number of channels that can be *active* at any time.

The default is the value specified for the `MaxChannels` attribute. For z/OS, the value must be in the range 1 through 9 999. For all other platforms, the value must be in the range 1 through 65 535.

To determine the value of this attribute, use the MQIA_ACTIVE_CHANNELS selector with the **MQINQ** call.

Related information:

Channel states

MaxChannels (MQLONG):

This attribute is the maximum number of channels that can be *current* (including server-connection channels with connected clients).

For z/OS, the value must be in the range 1 through 9 999, with a default value of 200. For all other platforms, the value must be in the range 1 through 65 535, with a default value of 100. A system that is busy serving connections from the network might need a higher number than the default setting. Determine the value that is correct for your environment, ideally by observing the behavior of your system during testing.

For platforms other than z/OS, the value for MaxChannels is set in the qm.ini file of respective queue managers.

To determine the value of this attribute, use the MQIA_MAX_CHANNELS selector with the **MQINQ** call.

Related information:
Channel states

MaxHandles (MQLONG):

This is the maximum number of open handles that any one task can use concurrently. Each successful MQOPEN call for a single queue (or for an object that is not a queue) uses one handle. That handle becomes available for reuse when the object is closed. However, when a distribution list is opened, each queue in the distribution list is allocated a separate handle, and so that MQOPEN call uses as many handles as there are queues in the distribution list. This must be taken into account when deciding on a suitable value for *MaxHandles*.

The MQPUT1 call performs an MQOPEN call as part of its processing; as a result, MQPUT1 uses as many handles as MQOPEN would, but the handles are used only for the duration of the MQPUT1 call itself.

On z/OS, *task* means a CICS task, an MVS task, or an IMS dependent region.

The value is in the range 1 through 999 999 999. The default value is determined by the environment:

- On z/OS, the default value is 100.
- In all other environments, the default value is 256.

To determine the value of this attribute, use the MQIA_MAX_HANDLES selector with the MQINQ call.

MaxMsgLength (MQLONG):

This is the length of the longest *physical* message that the queue manager can handle. However, because the *MaxMsgLength* queue-manager attribute can be set independently of the *MaxMsgLength* queue attribute, the longest physical message that can be placed on a queue is the lesser of those two values.

If the queue manager supports segmentation, an application can put a *logical* message that is longer than the lesser of the two *MaxMsgLength* attributes, but only if the application specifies the MQMF_SEGMENTATION_ALLOWED flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but usually resource constraints imposed by the operating system, or by the environment in which the application is running, result in a lower limit.

The lower limit for the *MaxMsgLength* attribute is 32 KB (32 768 bytes). The upper limit is 100 MB (104 857 600 bytes).

To determine the value of this attribute, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

MaxPriority (MQLONG):

This is the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to *MaxPriority* (highest).

To determine the value of this attribute, use the MQIA_MAX_PRIORITY selector with the MQINQ call.

MaxPropertiesLength (MQLONG):

This is used to control the size of the properties that can flow with a message. This includes both the property name in bytes and the size of the property value also in bytes.

To determine the value of this attribute, use the MQIA_MAX_PROPERTIES_LENGTH selector with the MQINQ call.

MaxUncommittedMsgs (MQLONG):

This is the maximum number of uncommitted messages that can exist within a unit of work. The number of uncommitted messages is the sum of the following since the start of the current unit of work:

- Messages put by the application with the MQPMO_SYNCPOINT option
- Messages retrieved by the application with the MQGMO_SYNCPOINT option
- Trigger messages and COA report messages generated by the queue manager for messages put with the MQPMO_SYNCPOINT option
- COD report messages generated by the queue manager for messages retrieved with the MQGMO_SYNCPOINT option

The following are *not* counted as uncommitted messages:

- Messages put or retrieved by the application outside a unit of work
- Trigger messages or COA/COD report messages generated by the queue manager as a result of messages put or retrieved outside a unit of work
- Expiration report messages generated by the queue manager (even if the call causing the expiration report message specified MQGMO_SYNCPOINT)
- Event messages generated by the queue manager (even if the call causing the event message specified MQPMO_SYNCPOINT or MQGMO_SYNCPOINT)

Note:

1. Exception report messages are generated by the Message Channel Agent (MCA), or by the application, and are treated in the same way as ordinary messages put or retrieved by the application.
2. When a message or segment is put with the MQPMO_SYNCPOINT option, the number of uncommitted messages is incremented by one regardless of how many physical messages actually result from the put. (More than one physical message might result if the queue manager must subdivide the message or segment.)
3. When a distribution list is put with the MQPMO_SYNCPOINT option, the number of uncommitted messages is incremented by one *for each physical message that is generated*. This can be as small as one, or as great as the number of destinations in the distribution list.

The lower limit for this attribute is 1; the upper limit is 999 999 999. The default value is 10000.

To determine the value of this attribute, use the MQIA_MAX_UNCOMMITTED_MSGS selector with the MQINQ call.

MQIAccounting (MQLONG):

This controls the collection of accounting information for MQI data.

The value is one of the following:

MQMON_ON
: Collect API accounting data.

MQMON_OFF
: Do not collect API accounting data. This is the default value.

If you set the queue manager attribute ACCTCONO to ENABLED, this value might be overridden for individual connections using the Options field in the MQCNO structure. Changes to this value are only effective for connections to the queue manager that occur after the change to the attribute.

This attribute is supported only on IBM i, UNIX systems, and Windows.

To determine the value of this attribute, use the MQIA_ACCOUNTING_MQI selector with the MQINQ call.

MQIStatistics (MQLONG):

This controls the collection of statistics monitoring information for the queue manager.

The value is one of the following:

MQMON_ON
Collect MQI statistics.

MQMON_OFF
Do not collect MQI statistics. This is the default value.

This attribute is supported only on IBM i, UNIX and Linux systems, and Windows.

To determine the value of this attribute, use the MQIA_STATISTICS_MQI selector with the MQINQ call.

MsgMarkBrowseInterval (MQLONG):

Time interval in milliseconds after which the queue manager can automatically remove the mark from browse messages.

This is a time interval (in milliseconds) after which the queue manager can automatically remove the mark from browse messages.

This attribute describes the time interval for which messages that have been marked as browsed by a call to MQGET, using the get message option MQGMO_MARK_BROWSE_CO_OP, are expected to remain marked as browsed.

The queue manager might automatically unmark browsed messages that have been marked as browsed for the cooperating set of handles when they have been marked for more than this approximate interval.

This does not affect the state of any message marked as browse, that was obtained by a call to MQGET, using the get message option MQGMO_MARK_BROWSE_HANDLE.

The value is not less than -1 and not greater than 999 999 999. The default value is 5000. A *MsgMarkBrowseInterval* of -1 represents an unlimited time interval. A *MsgMarkBrowseInterval* of 0 causes the queue manager to unmark the message immediately.

To determine the value of this attribute, use the MQIA_MSG_MARK_BROWSE_INTERVAL selector with the MQINQ call.

OutboundPortMax (MQLONG):

This is the highest port number in the range, defined by OutboundPortMin and OutboundPortMax, of port numbers to be used to bind outgoing channels.

The value is an integer in the range 0 through 65535, and must be equal to or greater than the OutboundPortMin value. The default value is 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_OUTBOUND_PORT_MAX selector with the MQINQ call.

OutboundPortMin (MQLONG):

This is the lowest port number in the range, defined by OutboundPortMin and OutboundPortMax, of port numbers to be used to bind outgoing channels.

The value is an integer in the range 0 through 65535, and must be equal to or less than the OutboundPortMax value. The default value is 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_OUTBOUND_PORT_MIN selector with the MQINQ call.

PerformanceEvent (MQLONG):

This controls whether performance-related events are generated. It is one of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_PERFORMANCE_EVENT selector with the MQINQ call.

Platform (MQLONG):

This indicates the operating system on which the queue manager is running:

MQPL_AIX
AIX (same value as MQPL_UNIX).

MQPL_MVS
z/OS (same value as MQPL_ZOS).

MQPL_NSK
HP Integrity NonStop Server.

MQPL_OS390
z/OS (same value as MQPL_ZOS).

MQPL_OS400
IBM i.

MQPL_UNIX
UNIX systems.

MQPL_WINDOWS_NT
Windows systems.

MQPL_ZOS
z/OS.

To determine the value of this attribute, use the MQIA_PLATFORM selector with the MQINQ call.

PubSubNPInputMsg (MQLONG):

Whether to discard or keep an undelivered input message.

The value is one of the following:

MQUNDELIVERED_DISCARD
Non-persistent input messages may be discarded if they cannot be processed.

This is the default value.

MQUNDELIVERED_KEEP
Non-persistent input messages will not be discarded if they cannot be processed. In this situation the queued publish/subscribe interface will continue to retry the process at appropriate intervals and does not continue processing subsequent messages.

To determine the value of this attribute, use the MQIA_PUBSUB_NP_MSG selector with the MQINQ call.

PubSubNPResponse (MQLONG):

Controls the behavior of undelivered response messages.

The value is one of the following:

MQUNDELIVERED_NORMAL
Non-persistent responses which cannot be placed on the reply queue are put on the dead letter queue, if they cannot be placed on the DLQ then they are discarded.

MQUNDELIVERED_SAFE
Non-persistent responses which cannot be placed on the reply queue are put on the dead letter queue. If the response cannot be set and cannot be placed on the DLQ then the queued publish/subscribe interface will roll back the current operation and then retry at appropriate intervals and does not continue processing subsequent messages.

MQUNDELIVERED_DISCARD
Non-persistent responses are not placed on the reply queue are discarded.

This is the default value for new queue managers.

MQUNDELIVERED_KEEP
Non-persistent responses are not placed on the dead letter queue or discarded. Instead, the queued publish/subscribe interface will back out the current operation and then retry it at appropriate intervals.

To determine the value of this attribute, use the MQIA_PUBSUB_NP_RESP selector with the MQINQ call.

Default value for migrated queue managers.

If the queue manager has been migrated from WebSphere MQ V6.0, the initial value of this attribute depends on the values of DiscardNonPersistentResponse and DLQNonPersistentResponse before migration, as shown in the following table.

		DLQNonPersistentResponse		
		Yes	**No**	**Not set**
DiscardNonPersistentResponse	**Yes**	MQUNDELIVERED_NORMAL	MQUNDELIVERED_DISCARD	MQUNDELIVERED_NORMAL
	No	MQUNDELIVERED_SAFE	MQUNDELIVERED_KEEP	MQUNDELIVERED_SAFE
	Not set	If SyncPointPersistent = No, MQUNDELIVERED_SAFE else MQUNDELIVERED_NORMAL	If SyncPointPersistent = No, MQUNDELIVERED_KEEP else MQUNDELIVERED_DISCARD	If SyncPointPersistent = No, MQUNDELIVERED_SAFE else MQUNDELIVERED_NORMAL

PubSubMaxMsgRetryCount (MQLONG):

The number of retries when processing a failed command message under syncpoint.

The value is one of the following:

0 - 999 999 999
The default value is 5.

To determine the value of this attribute, use the MQIA_PUBSUB_MAXMSG_RETRY_COUNT selector with the MQINQ call.

PubSubSyncPoint (MQLONG):

Whether only persistent messages or all messages are processed under syncpoint.

The value is one of the following:

MQSYNCPOINT_IFPER
This makes the queued publish/subscribe interface receive non-persistent messages outside syncpoint. If the daemon receives a publication outside syncpoint, the daemon forwards the publication to subscribers known to it outside syncpoint.

This is the default value.

MQSYNCPOINT_YES
This makes the queued publish/subscribe interface receive all messages under syncpoint.

To determine the value of this attribute, use the MQIA_PUBSUB_SYNC_PT selector with the MQINQ call.

PubSubMode (MQLONG):

Whether the publish/subscribe engine and the queued publish/subscribe interface are running, therefore allowing applications to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface.

The value is one of the following:

MQPSM_COMPAT
The publish/subscribe engine is running. It is therefore possible to publish/subscribe by using the application programming interface. The queued publish/subscribe interface is not running, therefore any message that is put to the queues that are monitored by the queued publish/subscribe interface is not acted on. This setting is used for compatibility with WebSphere Message Broker V6 or earlier versions using this queue manager, because it must read the same queues from which the queued publish/subscribe interface normally reads.

MQPSM_DISABLED
The publish/subscribe engine and the queued publish/subscribe interface are not running. It is therefore not possible to publish/subscribe by using the application programming interface. Any publish/subscribe messages that are put to the queues that are monitored by the queued publish/subscribe interface are not acted on.

MQPSM_ENABLED
The publish/subscribe engine and the queued publish/subscribe interface are running. It is therefore possible to publish/subscribe by using the application programming interface and the queues that are being monitored by the queued publish/subscribe interface. This is the queue manager's initial default value.

To determine the value of this attribute, use the MQIA_PUBSUB_MODE selector with the MQINQ call.

QMgrDesc (MQCHAR64):

Use this field for a commentary describing the queue manager. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, this field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

- On z/OS, the default value is the product name and version number.
- In all other environments, the default value is blanks.

To determine the value of this attribute, use the MQCA_Q_MGR_DESC selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_DESC_LENGTH.

QMgrIdentifier (MQCHAR48):

This is an internally-generated unique name for the queue manager.

To determine the value of this attribute, use the MQCA_Q_MGR_IDENTIFIER selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_IDENTIFIER_LENGTH.

This attribute is supported in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows, plus WebSphere MQ clients connected to these systems.

QMgrName (MQCHAR48):

This is the name of the local queue manager, that is, the name of the queue manager to which the application is connected.

The first 12 characters of the name are used to construct a unique message identifier (see MQMD - MsgId field). Queue managers that can intercommunicate must therefore have names that differ in the first 12 characters, in order for message identifiers to be unique in the queue-manager network.

On z/OS, the name is the same as the subsystem name, which is limited to 4 nonblank characters.

To determine the value of this attribute, use the MQCA_Q_MGR_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

QSGName (MQCHAR4):

This is the name of the queue-sharing group to which the local queue manager belongs. If the local queue manager does not belong to a queue-sharing group, the name is blank.

To determine the value of this attribute, use the MQCA_QSG_NAME selector with the MQINQ call. The length of this attribute is given by MQ_QSG_NAME_LENGTH.

This attribute is supported only on z/OS.

QueueAccounting (MQLONG):

This controls the collection of accounting information for queues.

The value is one of the following:

MQMON_NONE
Do not collect accounting data for queues, regardless of the setting of the queue accounting attribute ACCTQ. This is the default value.

MQMON_OFF
Do not collect accounting data for queues that specify QMGR in the ACCTQ queue attribute.

MQMON_ON
Collect accounting data for queues that specify QMGR in the ACCTQ queue attribute.

Changes to this value are only effective for connections to the queue manager that occur after the change to the attribute.

To determine the value of this attribute, use the MQIA_ACCOUNTING_Q selector with the MQINQ call.

QueueMonitoring (MQLONG):

This specifies the default setting for online monitoring of queues.

If the *QueueMonitoring* queue attribute is set to MQMON_Q_MGR, this attribute specifies the value which is assumed by the channel. The value can be:

MQMON_OFF
Online monitoring data collection is turned off. This is the queue manager's initial default value.

MQMON_NONE
Online monitoring data collection is turned off for queues regardless of the setting of their *QueueMonitoring* attribute.

MQMON_LOW
Online monitoring data collection is turned on, with a low ratio of data collection.

MQMON_MEDIUM
Online monitoring data collection is turned on, with a moderate ratio of data collection.

MQMON_HIGH
Online monitoring data collection is turned on, with a high ratio of data collection.

To determine the value of this attribute, use the MQIA_MONITORING_Q selector with the MQINQ call.

QueueStatistics (MQLONG):

This controls the collection of statistics data for queues.

It is one of the following values:

MQMON_NONE
Do not collect queue statistics for queues, regardless of the setting of the *QueueStatistics* queue attribute. This is the default value.

MQMON_OFF
Do not collect statistics data for queues that specify Queue Manager in the *QueueStatistics* queue attribute.

MQMON_ON
Collect statistics data for queues that specify Queue Manager in the *QueueStatistics* queue attribute.

To determine the value of this attribute, use the MQIA_STATISTICS_Q selector with the MQINQ call.

ReceiveTimeout (MQLONG):

This specifies how long a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state. It applies only to message channels and not to MQI channels.

The exact meaning of the ReceiveTimeout is altered by the value specified in ReceiveTimeoutType. ReceiveTimeoutType can be set to one of the following:

- MQRCVTIME_EQUAL - this value is the number in seconds for the channel to wait. Specify a value in the range 0 - 999999.
- MQRCVTIME_ADD - this value is the number in seconds to add to the negotiated HBINT, and it determines how long a channel waits. Specify a value in the range 1 - 999999.
- MQRCVTIME_MULTIPLY - this value is a multiplier to apply to the negotiated HBINT. Specify a value of 0 or a value in the range 2 - 99.

The default value is 0.

Set ReceiveTimeoutType to MQRCVTIME_MULTIPLY or MQRCVTIME_EQUAL, and ReceiveTimeout to 0, to stop a channel from timing out its wait to receive data from its partner.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_RECEIVE_TIMEOUT selector with the MQINQ call.

ReceiveTimeoutMin (MQLONG):

This is the minimum time, in seconds, that a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state.

It applies only to message channels, not to MQI channels. The value must be in the range 0 through 999999, with a default of 0.

If you use ReceiveTimeoutType to specify that the TCP/IP channel wait time is to be calculated relative to the negotiated value of HBINT, and the resultant value is less than the value of this parameter, this value is used instead.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_RECEIVE_TIMEOUT_MIN selector with the MQINQ call.

ReceiveTimeoutType (MQLONG):

This is the qualifier, applied to ReceiveTimeout to define how long a TCP/IP channel waits to receive data, including heartbeats, from its partner, before returning to the inactive state. It applies only to message channels, not to MQI channels.

The value is one of the following:

MQRCVTIME_MULTIPLY
ReceiveTimeout is a multiplier to apply to the negotiated HBINT value to determine how long a channel waits. This is the default value.

MQRCVTIME_ADD
ReceiveTimeout is a value, in seconds, to add to the negotiated HBINT value to determine how long a channel waits.

MQRCVTIME_EQUAL
ReceiveTimeout is a value, in seconds, that the channel waits.

To stop a channel timing out its wait to receive data from its partner, set ReceiveTimeoutType to MQRCVTIME_MULTIPLY or MQRCVTIME_EQUAL, and ReceiveTimeout to 0.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_RECEIVE_TIMEOUT_TYPE selector with the MQINQ call.

RemoteEvent (MQLONG):

This controls whether remote error events are generated. It is one of the following values:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_REMOTE_EVENT selector with the MQINQ call.

RepositoryName (MQCHAR48):

This is the name of a cluster for which this queue manager provides a repository-manager service. If the queue manager provides this service for more than one cluster, *RepositoryNamelist* specifies the name of a namelist object that identifies the clusters, and *RepositoryName* is blank. At least one of *RepositoryName* and *RepositoryNamelist* must be blank.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS.

To determine the value of this attribute, use the MQCA_REPOSITORY_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

RepositoryNamelist (MQCHAR48):

This is the name of a namelist object that contains the names of clusters for which this queue manager provides a repository-manager service. If the queue manager provides this service for only one cluster, the namelist object contains only one name. Alternatively, *RepositoryName* can be used to specify the name of the cluster, in which case *RepositoryNamelist* is blank. At least one of *RepositoryName* and *RepositoryNamelist* must be blank.

This attribute is supported only on AIX, HP-UX, IBM i, Linux, Solaris, Windows, and z/OS.

To determine the value of this attribute, use the MQCA_REPOSITORY_NAMELIST selector with the MQINQ call. The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

ScyCase(MQCHAR8):

Specifies whether the queue manager supports security profile names in mixed case, or in uppercase only.

The value is one of the following:

MQSCYC_UPPER
Security profile names must be in uppercase.

MQSCYC_MIXED
Security profile names can be in uppercase or in mixed case.

Changes to this attribute take effect when a Refresh Security command is run with *SecurityType(MQSECTYPE_CLASSES)* specified.

This attribute is supported only on z/OS.

To determine the value of this attribute, use the MQIA_SECURITY_CASE selector with the MQINQ call.

SharedQMgrName (MQLONG):

This specifies whether the *ObjectQmgrName* should be used or treated as the local queue manager on an MQOPEN call, for a shared queue, when the *ObjectQmgrName* is that of another queue manager in the queue-sharing group.

The value can be:

MQSQQM_USE
ObjectQmgrName is used and the appropriate transmission queue is opened.

MQSQQM_IGNORE
If the target queue is shared, and the *ObjectQmgrName* is that of a queue manager in the same queue-sharing group, the open is performed locally.

This attribute is valid only on z/OS.

To determine the value of this attribute, use the MQIA_SHARED_Q_Q_MGR_NAME selector with the MQINQ call.

SPLCAP:

Indicates whether security capabilities of WebSphere MQ Advanced Message Security are available for a queue manager.

MQCAP_SUPPORTED
This is the default value if the WebSphere MQ AMS component is installed for the installation that the queue manager is running under.

MQCAP_NOT_SUPPORTED

SSLEvent (MQLONG):

This specifies whether SSL events are generated.

It is one of the following values:

MQEVR_ENABLED
Generate SSL events, as follows:
MQRC_CHANNEL_SSL_ERROR

MQEVR_DISABLED
Do not generate SSL events; this is the default value.

To determine the value of this attribute, use the MQIA_SSL_EVENT selector with the MQINQ call.

SSLFIPSRequired (MQLONG):

This lets you specify that only FIPS-certified algorithms are to be used if the cryptography is executed in WebSphere MQ, rather than in cryptographic hardware. If cryptographic hardware is configured, the cryptography modules used are those modules provided by the hardware product; these modules might or might not be FIPS-certified to a particular level depending on the hardware product in use.

The value is one of the following values:

MQSSL_FIPS_NO
Use any CipherSpec supported on the platform in use. This value is the default value.

MQSSL_FIPS_YES
Use only FIPS-certified cryptographic algorithms in the CipherSpecs allowed on all SSL connections from and to this queue manager.

This parameter is valid only on UNIX, Linux, Windows, and z/OS platforms.

To determine the value of this attribute, use the MQIA_SSL_FIPS_REQUIRED selector with the MQINQ call.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

SSLKeyResetCount (MQLONG):

This specifies when SSL channel message channel agents (MCAs) that initiate communication reset the secret key used for encryption on the channel.

The value represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

The value is a number in the range 0 through 999 999 999, with a default value of 0. If you specify an SSL/TLS secret key reset count in the range 1 byte through 32 KB, SSL/TLS channels will use a secret key reset count of 32 KB. This is to avoid the processing cost of excessive key resets which would occur for small SSL/TLS secret key reset values.

The secret key is renegotiated when the total number of unencrypted bytes sent and received by the initiating channel MCA exceeds the specified value, or if channel heartbeats are enabled before data is sent or received following a channel heartbeat, whichever occurs first.

The count of bytes sent and received for renegotiation includes control information sent and received by the channel MCA and is reset whenever a renegotiation occurs.

Use a value of 0 to indicate that secret keys are never renegotiated.

To determine the value of this attribute, use the MQIA_SSL_RESET_COUNT selector with the MQINQ call.

StartStopEvent (MQLONG):

This controls whether start and stop events are generated. The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_START_STOP_EVENT selector with the MQINQ call.

StatisticsInterval (MQLONG):

This specifies how often (in seconds) to write statistics monitoring data to the monitoring queue.

The value is an integer in the range 0 to 604800, with a default value of 1800 (30 minutes).

To determine the value of this attribute, use the MQIA_STATISTICS_INTERVAL selector with the MQINQ call.

SyncPoint (MQLONG):

This indicates whether the local queue manager supports units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls.

MQSP_AVAILABLE
Units of work and syncpointing available.

MQSP_NOT_AVAILABLE
Units of work and syncpointing not available.

- On z/OS this value is never returned.

To determine the value of this attribute, use the MQIA_SYNCPOINT selector with the MQINQ call.

TCPChannels (MQLONG):

This is the maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol.

The value must be in the range 0 through 9999, with a default value of 200. If you specify 0, TCP/IP is not used.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_CHANNELS selector with the MQINQ call.

TCPKeepAlive (MQLONG):

This specifies whether to use TCP KEEPALIVE to check that the other end of the connection is still available. If it is not available, the channel is closed.

The value is one of the following:

MQTCPKEEP_YES
Use TCP KEEPALIVE as specified in the TCP profile configuration data set. If you specify the channel attribute KeepAliveInterval (KAINT), the value to which it is set is used.

MQTCPKEEP_NO
Do not use TCP KEEPALIVE. This is the default value.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_KEEP_ALIVE selector with the MQINQ call.

TCPName (MQCHAR8):

This is the name of either the only or default TCP/IP system that you are using, depending on the value of TCPStackType. The default value is TCPIP.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQCA_TCP_NAME selector with the MQINQ call. The length of this attribute is given by MQ_TCP_NAME_LENGTH.

TCPStackType (MQLONG):

This specifies whether the channel initiator can use only the TCP/IP address space specified in TCPName, or can optionally bind to any selected TCP/IP address

The value is one of the following:

MQTCPSTACK_SINGLE
The channel initiator can use only the TCP/IP address spaces named in TCPName. This is the default value.

MQTCPSTACK_MULTIPLE
The channel initiator can use any TCP/IP address space available to it. It defaults to the one specified in TCPName if no other is specified for a channel or listener.

This attribute is supported on z/OS only.

To determine the value of this attribute, use the MQIA_TCP_STACK_TYPE selector with the MQINQ call.

TraceRouteRecording (MQLONG):

This controls the recording of trace- route information.

The value is one of the following:

MQRECORDING_DISABLED
No appending to trace- route messages allowed.

MQRECORDING_Q
Put trace- route messages to fixed named queue.

MQRECORDING_MSG
Put trace- route messages to a queue determined using the message itself. This is the default value

To determine the value of this attribute, use the MQIA_TRACE_ROUTE_RECORDING selector with the MQINQ call.

TriggerInterval (MQLONG):

This is a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the *TriggerType* is MQTT_FIRST. In this case trigger messages are usually generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

For more information on triggering, see Triggering channels.

The value is not less than 0 and not greater than 999 999 999. The default value is 999 999 999.

To determine the value of this attribute, use the MQIA_TRIGGER_INTERVAL selector with the MQINQ call.

TriggerInterval (MQLONG):

This is a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the *TriggerType* is MQTT_FIRST. In this case trigger messages are usually generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

For more information on triggering, see Triggering channels.

The value is not less than 0 and not greater than 999 999 999. The default value is 999 999 999.

To determine the value of this attribute, use the MQIA_TRIGGER_INTERVAL selector with the MQINQ call.

Version (MQCFST):

This is the version of the WebSphere MQ code as VVRRMMFF, where:

VV - Version

RR - Release

MM - Maintenance level

FF - Fix level

XrCapability(MQLONG):

This controls whether WebSphere MQ Telemetry commands are supported by the queue manager.

The value is one of the following:

MQCAP_SUPPORTED
WebSphere MQ Telemetry component installed and Telemetry commands are supported.

MQCAP_NOT_SUPPORTED
webSphere MQ Telemetry component not installed.

This attribute is supported only on IBM i, Unix systems, and Windows.

To determine the value of this attribute, use the MQIA_XR_CAPABILITY selector with the MQINQ call.

Attributes for queues:

There are five types of queue definition. Some queue attributes apply to all types of queue; other queue attributes apply only to certain types of queue.

Types of queue

The queue manager supports the following types of queue definition:

Local queue

You can store messages on a local queue. On z/OS you can make it a shared or private queue.

A queue is known to a program as *local* if it is owned by the queue manager to which the program is connected. You can get messages from, and put messages on, local queues.

The queue definition object holds the definition information of the queue as well as the physical messages put on the queue.

Local queue manager queue

The queue exists on the local queue manager. The queue is known as a private queue on z/OS.

Shared queue (z/OS only)

The queue exists in a shared repository that is accessible to all the queue managers that belong to the queue-sharing group that owns the shared repository.

Applications connected to any queue manager in the queue-sharing group can place messages on and remove messages from queues of this type. Such queues are effectively the same as local queues. The value of the *QType* queue attribute is MQQT_LOCAL.

Applications connected to the local queue manager can place messages on and remove messages from queues of this type. The value of the *QType* queue attribute is MQQT_LOCAL.

Cluster queue

You can store messages on a cluster queue on the queue manager where it is defined. A cluster queue is a queue that is hosted by a cluster queue manager and made available to other queue managers in the cluster. The value of the *QType* queue attribute is MQQT_CLUSTER.

A cluster queue definition is advertised to other queue managers in the cluster. The other queue managers in the cluster can put messages to a cluster queue without needing a corresponding remote-queue definition. A cluster queue can be advertised in more than one cluster by using a cluster namelist.

When a queue is advertised, any queue manager in the cluster can put messages to it. To put a message, the queue manager must find out, from the full repositories, where the queue is hosted. Then it adds some routing information to the message and puts the message on a cluster transmission queue.

Except on z/OS, a queue manager can store messages for other queue managers in a cluster on multiple transmission queues. You can configure a queue manager to store messages on multiple cluster transmission queues in two different ways. If you set the queue manager attribute DEFCLXQ to CHANNEL, a different cluster transmission queue is created automatically from SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE for each cluster-sender channel. If you set the CLCHNAME transmission queue option to match one or more cluster-senders channel, the queue manager can store messages for the matching channels on that transmission queue.

A cluster queue can be a queue that is shared by members of a queue-sharing group in IBM WebSphere MQ for z/OS.

Remote queue

A remote queue is not a physical queue; it is the local definition of a queue that exists on a remote queue manager. The local definition of the remote queue contains information that tells the local queue manager how to route messages to the remote queue manager.

Applications connected to the local queue manager can place messages on queues of this type; the messages are placed on the local transmission queue used to route messages to the remote queue manager. Applications cannot remove messages from remote queues. The value of the *QType* queue attribute is MQQT_REMOTE.

You can also use a remote queue definition for:

- Reply-queue aliasing

  In this case the name of the definition is the name of a reply-to queue. For more information, see Reply-to queue aliases and clusters.

- Queue-manager aliasing

  In this case the name of the definition is an alias for a queue manager, and not the name of a queue. For more information, see Queue manager aliases and clusters.

Alias queue

This is not a physical queue; it is an alternative name for a local queue, a shared queue, a cluster queue, or a remote queue. The name of the queue to which the alias resolves is part of the definition of the alias queue.

Applications connected to the local queue manager can place messages on queues of this type; the messages are placed on the queue to which the alias resolves. Applications can remove messages from queues of this type if the alias resolves to a local queue, a shared queue, or a cluster queue that has a local instance. The value of the *QType* queue attribute is MQQT_ALIAS.

Model queue

This is not a physical queue; it is a set of queue attributes from which a local queue can be created.

Messages cannot be stored on queues of this type.

Queue attributes

Some queue attributes apply to all types of queue; other queue attributes apply only to certain types of queue. The types of queue to which an attribute applies are shown in Table 220 on page 2137 and subsequent tables.

Table 220 on page 2137 summarizes the attributes that are specific to queues. The attributes are described in alphabetical order.

Note: The names of the attributes shown in this section are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) commands for details.

Table 220. Attributes for queues. The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
AlterationDate	Date when definition was last changed	✓		✓	✓	
AlterationTime	Time when definition was last changed	✓		✓	✓	
BackoutRequeueQName	Excessive backout requeue queue name	✓	✓			
BackoutThreshold	Backout threshold	✓	✓			
BaseQName	Queue name to which alias resolves			✓		
CFStrucName	Coupling-facility structure name	✓	✓			
CLCHNAME	Cluster-sender channel names	✓	✓			
ClusterName	Name of cluster to which queue belongs	✓		✓	✓	✓
ClusterNamelist	Name of namelist object containing names of clusters to which queue belongs	✓		✓	✓	
CLWLQueuePriority	Cluster workload queue priority	✓		✓	✓	✓
CLWLQueueRank	Cluster workload queue rank	✓		✓	✓	✓
CLWLUseQ	Use remote queue	✓				
CreationDate	Date that the queue was created	✓				
CreationTime	Time that the queue was created	✓				
CurrentQDepth	Current queue depth	✓				
DefaultPutResponse	Default put response	✓	✓	✓	✓	
DefBind	Default binding	✓		✓	✓	✓
DefinitionType attribute	Queue definition type	✓	✓			
DefInputOpenOption	Default input open option	✓	✓			
DefPersistence	Default message persistence	✓	✓	✓	✓	✓
DefPriority	Default message priority	✓	✓	✓	✓	✓
DefReadAhead	Default read ahead	✓	✓	✓		

Table 220. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
DistLists	Distribution list support	✓	✓			
HardenGetBackout	Whether to maintain an accurate backout count	✓	✓			
IndexType	Index type	✓	✓			
InhibitGet	Whether get operations for the queue are allowed	✓	✓	✓		
InhibitPut	Whether put operations for the queue are allowed	✓	✓	✓	✓	✓
InitiationQName	Name of initiation queue	✓	✓			
MaxMsgLength	Maximum message length in bytes	✓	✓			
MaxQDepth	Maximum queue depth	✓	✓			
MsgDeliverySequence attribute	Message delivery sequence	✓	✓			
NonPersistentMessage Class	Reliability goal for non-persistent messages	✓	✓			
OpenInputCount	Number of opens for input	✓				
OpenOutputCount	Number of opens for output	✓				
PropertyControl	Property control	✓	✓	✓		
ProcessName	Process name	✓	✓			
QDepthHighEvent attribute	Whether Queue Depth High events are generated	✓	✓			
QDepthHighLimit	High limit for queue depth	✓	✓			
QDepthLowEvent attribute	Whether Queue Depth Low events are generated	✓	✓			
QDepthLowLimit attribute	Low limit for queue depth	✓	✓			
QDepthMaxEvent	Whether Queue Full events are generated	✓	✓			
QDesc	Queue description	✓	✓	✓	✓	✓
QName	Queue name	✓		✓	✓	✓
QServiceInterval	Target for queue service interval	✓	✓			

Table 220. Attributes for queues (continued). The columns apply as follows:

- The column for local queues applies also to shared queues.
- The column for model queues indicates which attributes are inherited by the local queue created from the model queue.
- The column for cluster queues indicates the attributes that can be inquired when the cluster queue is opened for inquire alone, or for inquire and output. If the cluster queue is opened for inquire plus one or more of input, browse, or set, the column for local queues applies instead.

Attribute	Description	Local	Model	Alias	Remote	Cluster
QServiceIntervalEvent attribute	Whether Service Interval High or Service Interval OK events are generated	✓	✓			
QSGDisp attribute	Queue-sharing group disposition	✓		✓	✓	
QueueAccounting	Queue accounting data collection	✓	✓	✓	✓	✓
QueueMonitoring	Online monitoring data for queues	✓	✓			
QueueStatistics	Queue statistics data collection	✓	✓	✓	✓	✓
QType	Queue type	✓		✓	✓	✓
RemoteQMgrName	Name of remote queue manager				✓	
RemoteQName	Name of remote queue				✓	
RetentionInterval	Retention interval	✓	✓			
Scope	Whether an entry for the queue also exists in a cell directory	✓		✓	✓	
Shareability	Queue shareability	✓	✓			
StorageClass	Storage class for queue	✓	✓			
TriggerControl	Trigger control	✓	✓			
TriggerData	Trigger data	✓	✓			
TriggerDepth	Trigger depth	✓	✓			
TriggerMsgPriority	Threshold message priority for triggers	✓	✓			
TriggerType	Trigger type	✓	✓			
Usage attribute	Queue usage	✓	✓			
XmitQName	Transmission queue name				✓	

Related information:
Cluster queues
Local queues

AlterationDate (MQCHAR12):

Date when definition was last changed.

Local	Model	Alias	Remote	Cluster
X		X	X	

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes (for example, `1992-09-23¬¬`, where ¬¬ represents two blank characters).

The values of certain attributes (for example, *CurrentQDepth*) change as the queue manager operates. Changes to these attributes do not affect *AlterationDate*.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8):

Time when definition was last changed.

Local	Model	Alias	Remote	Cluster
X		X	X	

This is the time when the definition was last changed. The format of the time is `HH.MM.SS` using the 24-hour clock, with a leading zero if the hour is less than 10 (for example `09.10.20`).

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.

The values of certain attributes (for example, *CurrentQDepth*) change as the queue manager operates. Changes to these attributes do not affect *AlterationTime*.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

BackoutRequeueQName (MQCHAR48):

This is the excessive backout requeue queue name. Apart from allowing its value to be queried, the queue manager takes no action based on the value of this attribute.

Local	Model	Alias	Remote	Cluster
X	X			

Applications running inside WebSphere Application Server and those that use the WebSphere MQ Application Server Facilities use this attribute to determine where messages that have been backed out should go. For all other applications, the queue manager takes no action based on the value of the attribute.

WebSphere MQ classes for JMS uses this attribute to determine where to transfer a message that has already been backed out the maximum number of times as specified by the *BackoutThreshold* attribute.

To determine the value of this attribute, use the MQCA_BACKOUT_REQ_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

BackoutThreshold (MQLONG):

This is the backout threshold. Apart from allowing its value to be queried, the queue manager takes no action based on the value of this attribute.

Local	Model	Alias	Remote	Cluster
X	X			

Applications running inside of WebSphere Application Server and those that use the WebSphere MQ Application Server Facilities will use this attribute to determine if a message should be backed out. For all other applications, the queue manager takes no action based on the value of the attribute.

WebSphere MQ classes for JMS uses this attribute to determine how many times to allow a message to be backed out before transferring the message to the queue specified by the *BackoutRequeueQName* attribute.

To determine the value of this attribute, use the MQIA_BACKOUT_THRESHOLD selector with the MQINQ call.

BaseQName (MQCHAR48):

This is the name of a queue that is defined to the local queue manager.

Local	Model	Alias	Remote	Cluster
		X		

(For more information on queue names, see MQOD - ObjectName field.) The queue is one of the following types:

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_CLUSTER
Cluster queue.

To determine the value of this attribute, use the MQCA_BASE_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

BaseType (MQCFIN):

The type of object to which the alias resolves.

Local	Model	Alias	Remote	Cluster
		X		

It is one of the following values:

MQOT_Q
Base object type is a queue

MQOT_TOPIC
Base object type is a topic

CFStrucName (MQCHAR12):

This is the name of the coupling-facility structure where the messages on the queue are stored. The first character of the name is in the range A through Z, and the remaining characters are in the range A through Z, 0 through 9, or blank.

Local	Model	Alias	Remote	Cluster
X	X			

To get the full name of the structure in the coupling facility, suffix the value of the *QSGName* queue-manager attribute with the value of the *CFStrucName* queue attribute.

This attribute applies only to shared queues; it is ignored if *QSGDisp* does not have the value MQQSGD_SHARED.

To determine the value of this attribute, use the MQCA_CF_STRUC_NAME selector with the MQINQ call. The length of this attribute is given by MQ_CF_STRUC_NAME_LENGTH.

This attribute is supported only on z/OS.

ClusterChannelName (MQCHAR20):

ClusterChannelName is the generic name of the cluster-sender channels that use this queue as a transmission queue. The attribute specifies which cluster-sender channels send messages to a cluster-receiver channel from this cluster transmission queue. ClusterChannelName is not supported on z/OS.

Local	Model	Alias	Remote	Cluster
✓	✓			

The default queue manager configuration is for all cluster-sender channels to send messages from a single transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. The default configuration can be changed by modified by changing the queue manager attribute, DefClusterXmitQueueType. The default value of the attribute is SCTQ. You can change the value to CHANNEL. If you set the DefClusterXmitQueueType attribute to CHANNEL, each cluster-sender channel defaults to using a specific cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

You can also set the transmission queue attribute ClusterChannelName attribute to a cluster-sender channel manually. Messages that are destined for the queue manager connected by the cluster-sender channel are stored in the transmission queue that identifies the cluster-sender channel. They are not stored in the default cluster transmission queue. If you set the ClusterChannelName attribute to blanks, the channel switches to the default cluster transmission queue when the channel restarts. The default queue is either SYSTEM.CLUSTER.TRANSMIT.*ChannelName* or SYSTEM.CLUSTER.TRANSMIT.QUEUE, depending on the value of the queue manager DefClusterXmitQueueType attribute.

By specifying asterisks, "*", in ClusterChannelName, you can associate a transmission queue with a set of cluster-sender channels. The asterisks can be at the beginning, end, or any number of places in the middle of the channel name string. ClusterChannelName is limited to a length of 20 characters: MQ_CHANNEL_NAME_LENGTH.

ClusterName (MQCHAR48):

This is the name of the cluster to which the queue belongs.

Local	Model	Alias	Remote	Cluster
X		X	X	X

If the queue belongs to more than one cluster, *ClusterNamelist* specifies the name of a namelist object that identifies the clusters, and *ClusterName* is blank. At least one of *ClusterName* and *ClusterNamelist* must be blank.

To determine the value of this attribute, use the MQCA_CLUSTER_NAME selector with the MQINQ call. The length of this attribute is given by MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCHAR48):

This is the name of a namelist object that contains the names of clusters to which this queue belongs.

Local	Model	Alias	Remote	Cluster
X		X	X	

If the queue belongs to only one cluster, the namelist object contains only one name. Alternatively, *ClusterName* can be used to specify the name of the cluster, in which case *ClusterNamelist* is blank. At least one of *ClusterName* and *ClusterNamelist* must be blank.

To determine the value of this attribute, use the MQCA_CLUSTER_NAMELIST selector with the MQINQ call. The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

CLWLQueuePriority (MQLONG):

This is the cluster workload queue priority, a value in the range 0 through 9 representing the priority of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

For more information, see Cluster queues.

To determine the value of this attribute, use the MQIA_CLWL_Q_PRIORITY selector with the MQINQ call.

CLWLQueueRank (MQLONG):

This is the cluster workload queue rank, a value in the range 0 through 9 representing the rank of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

For more information, see Cluster queues.

To determine the value of this attribute, use the MQIA_CLWL_Q_RANK selector with the MQINQ call.

CLWLUseQ (MQLONG):

This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance. If the put originates from a cluster channel, this attribute does not apply.

Local	Model	Alias	Remote	Cluster
X				

The value is one of the following:

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

MQCLWL_USEQ_AS_Q_MGR
Inherit definition from queue manager's MQIA_CLWL_USEQ.

For more information, see Cluster queues.

To determine the value of this attribute, use the MQCA_CLWL_USEQ selector with the MQINQ call. The length of this attribute is given by MQ_CLWL_USEQ_LENGTH.

CreationDate (MQCHAR12):

This is the date when the queue was created.

Local	Model	Alias	Remote	Cluster
X				

The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes (for example, `2013-09-23¬¬`, where ¬¬ represents 2 blank characters).

- On IBM i, the creation date of a queue can differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the MQCA_CREATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_CREATION_DATE_LENGTH.

CreationTime (MQCHAR8):

This is the time when the queue was created.

Local	Model	Alias	Remote	Cluster
X				

The format of the time is `HH.MM.SS` using the 24-hour clock, with a leading zero if the hour is less than 10 (for example `09.10.20`).

- On z/OS, the time is Greenwich Mean Time (GMT), subject to the system clock being set accurately to GMT.
- In other environments, the time is local time.
- On IBM i, the creation time of a queue can differ from that of the underlying operating system entity (file or userspace) that represents the queue.

To determine the value of this attribute, use the MQCA_CREATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_CREATION_TIME_LENGTH.

CurrentQDepth (MQLONG):

This is the number of messages currently on the queue.

Local	Model	Alias	Remote	Cluster
X				

It is incremented during an MQPUT call, and during backout of an MQGET call. It is decremented during a nonbrowse MQGET call, and during backout of an MQPUT call. The effect of this is that the count includes messages that have been put on the queue within a unit of work, but that have not yet been committed, even though they are not eligible to be retrieved by the MQGET call. Similarly, it excludes messages that have been retrieved within a unit of work using the MQGET call, but that have yet to be committed.

The count also includes messages that have passed their expiry time but have not yet been discarded, although these messages are not eligible to be retrieved. See MQMD - Expiry field for more information.

Unit-of-work processing and the segmentation of messages can both cause *CurrentQDepth* to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages; *all* messages on the queue can be retrieved using the MQGET call in the normal way.

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_CURRENT_Q_DEPTH selector with the MQINQ call.

DefaultPutResponse (MQLONG):

Specifies the type of response to be used for put operations to the queue when an application specifies MQPMO_RESPONSE_AS_Q_DEF.

Local	Model	Alias	Remote	Cluster
X	X	X	X	

It is one of the following values:

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

DefBind (MQLONG):

This is the default binding that is used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call and the queue is a cluster queue.

Local	Model	Alias	Remote	Cluster
X		X	X	X

The value is one of the following:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance. Because this value is new in IBM WebSphere MQ Version 7.1, it must not be used if any of the applications opening this queue are connecting to IBM WebSphere MQ Version 7.0.1 or earlier queue managers.

To determine the value of this attribute, use the MQIA_DEF_BIND selector with the MQINQ call.

DefinitionType (MQLONG):

This indicates how the queue was defined.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQQDT_PREDEFINED
The queue is a permanent queue created by the system administrator; only the system administrator can delete it.

Predefined queues are created using the DEFINE MQSC command, and can be deleted only by using the DELETE MQSC command. Predefined queues cannot be created from model queues.

Commands can be issued either by an operator, or by an authorized user sending a command message to the command input queue (see CommandInputQName attribute for more information).

MQQDT_PERMANENT_DYNAMIC
The queue is a permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_PERMANENT_DYNAMIC for the *DefinitionType* attribute.

This type of queue can be deleted using the MQCLOSE call. See "MQCLOSE - Close object" on page 1950 for more details.

The value of the *QSGDisp* attribute for a permanent dynamic queue is MQQSGD_Q_MGR.

MQQDT_TEMPORARY_DYNAMIC
The queue is a temporary queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_TEMPORARY_DYNAMIC for the *DefinitionType* attribute.

This type of queue is deleted automatically by the MQCLOSE call when it is closed by the application that created it.

The value of the *QSGDisp* attribute for a temporary dynamic queue is MQQSGD_Q_MGR.

MQQDT_SHARED_DYNAMIC
The queue is a shared permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_SHARED_DYNAMIC for the *DefinitionType* attribute.

This type of queue can be deleted using the MQCLOSE call. See "MQCLOSE - Close object" on page 1950 for more details.

The value of the *QSGDisp* attribute for a shared dynamic queue is MQQSGD_SHARED.

This attribute in a model queue definition does not indicate how the model queue was defined, because model queues are always predefined. Instead, the value of this attribute in the model queue is used to determine the *DefinitionType* of each of the dynamic queues created from the model queue definition using the MQOPEN call.

To determine the value of this attribute, use the MQIA_DEFINITION_TYPE selector with the MQINQ call.

DefInputOpenOption (MQLONG):

This is the default way in which to open the queue for input.

Local	Model	Alias	Remote	Cluster
X	X			

It applies if the MQOO_INPUT_AS_Q_DEF option is specified on the MQOPEN call when the queue is opened. The value is one of the following:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type (MQOO_INPUT_SHARED or MQOO_INPUT_EXCLUSIVE).

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

To determine the value of this attribute, use the MQIA_DEF_INPUT_OPEN_OPTION selector with the MQINQ call.

DefPersistence (MQLONG):

This is the default persistence of messages on the queue. It applies if MQPER_PERSISTENCE_AS_Q_DEF is specified in the message descriptor when the message is put.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

If there is more than one definition in the queue-name resolution path, the default persistence is taken from the value of this attribute in the *first* definition in the path at the time of the MQPUT or MQPUT1 call. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The value is one of the following:

MQPER_PERSISTENT
The message survives system failures and queue manager restarts. Persistent messages cannot be placed on:

- Temporary dynamic queues
- Shared queues that map to a CFSTRUCT object at CFLEVEL(2) or below, or where the CFSTRUCT object is defined as RECOVER(NO).

Persistent messages can be placed on permanent dynamic queues, and predefined queues.

MQPER_NOT_PERSISTENT
The message does not normally survive system failures or queue manager restarts. This applies even if an intact copy of the message is found on auxiliary storage during a queue manager restart.

In the case of shared queues, nonpersistent messages *do* survive restarts of queue managers in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

Both persistent and nonpersistent messages can exist on the same queue.

To determine the value of this attribute, use the MQIA_DEF_PERSISTENCE selector with the MQINQ call.

DefPriority (MQLONG):

This is the default priority for messages on the queue. This applies if MQPRI_PRIORITY_AS_Q_DEF is specified in the message descriptor when the message is put on the queue.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

If there is more than one definition in the queue-name resolution path, the default priority for the message is taken from the value of this attribute in the *first* definition in the path at the time of the put operation. This could be:

- An alias queue
- A local queue
- A local definition of a remote queue
- A queue-manager alias
- A transmission queue (for example, the *DefXmitQName* queue)

The way in which a message is placed on a queue depends on the value of the queue's *MsgDeliverySequence* attribute:

- If the *MsgDeliverySequence* attribute is MQMDS_PRIORITY, the logical position at which a message is placed on the queue depends on the value of the *Priority* field in the message descriptor.
- If the *MsgDeliverySequence* attribute is MQMDS_FIFO, messages are placed on the queue as though they had a priority equal to the *DefPriority* of the resolved queue, regardless of the value of the *Priority* field in the message descriptor. However, the *Priority* field retains the value specified by the application that put the message. See MsgDeliverySequence attribute for more information.

Priorities are in the range zero (lowest) through *MaxPriority* (highest); see MaxPriority attribute.

To determine the value of this attribute, use the MQIA_DEF_PRIORITY selector with the MQINQ call.

DefReadAhead (MQLONG):

Specifies the default read ahead behavior for non-persistent messages delivered to the client.

Local	Model	Alias	Remote	Cluster
X	X	X		

DefReadAhead can be set to one of the following values::

MQREADA_NO
Non-persistent messages are not sent ahead to the client before an applications requests them. A maximum of one non-persistent message can be lost if the client ends abnormally.

MQREADA_YES
Non-persistent messages are sent ahead to the client before an application requests them. Non-persistent messages can be lost if the client ends abnormally or if the client does not consume all the messages it is sent.

MQREADA_DISABLED
Read ahead of non-persistent messages in not enabled for this queue. Messages are not sent ahead to the client regardless of whether read ahead is requested by the client application.

To determine the value of this attribute, use the MQIA_DEF_READ_AHEAD selector with the MQINQ call.

DefPResp (MQLONG):

The default put response type (DEFPRESP) attribute defines the value used by applications when the PutResponseType within MQPMO has been set to MQPMO_RESPONSE_AS_Q_DEF. This attribute is valid for all queue types.

Local	Model	Alias	Remote	Cluster
Yes	Yes	Yes	Yes	Yes

The value is one of the following:

SYNC The put operation is issued synchronously returning a response.

ASYNC
The put operation is issued asynchronously, returning a subset of MQMD fields.

To determine the value of this attribute, use the MQIA_DEF_PUT_RESPONSE_TYPE selector with the MQINQ call.

DistLists (MQLONG):

This indicates whether distribution-list messages can be placed on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

A message channel agent (MCA) sets the attribute to inform the local queue manager whether the queue manager at the other end of the channel supports distribution lists. This latter queue manager (called the *partnering* queue manager) is the one that next receives the message, after it has been removed from the local transmission queue by a sending MCA.

The sending MCA sets the attribute whenever it establishes a connection to the receiving MCA on the partnering queue manager. In this way, the sending MCA can cause the local queue manager to place on the transmission queue only messages that the partnering queue manager can process correctly.

This attribute is primarily for use with transmission queues, but the processing described is performed regardless of the usage defined for the queue (see Usage attribute).

The value is one of the following:

MQDL_SUPPORTED
Distribution-list messages can be stored on the queue, and transmitted to the partnering queue manager in that form. This reduces the amount of processing required to send the message to multiple destinations.

MQDL_NOT_SUPPORTED
Distribution-list messages cannot be stored on the queue, because the partnering queue manager does not support distribution lists. If an application puts a distribution-list message, and that message is to be placed on this queue, the queue manager splits the distribution-list message and places the individual messages on the queue instead. This increases the amount of processing required to send the message to multiple destinations, but ensures that the messages are processed correctly by the partnering queue manager.

To determine the value of this attribute, use the MQIA_DIST_LISTS selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

This attribute is not supported on z/OS.

HardenGetBackout (MQLONG):

For each message, a count is kept of the number of times that the message is retrieved by an MQGET call within a unit of work, and that unit of work subsequently backed out.

Local	Model	Alias	Remote	Cluster
X	X			

This count is available in the *BackoutCount* field in the message descriptor after the MQGET call has completed.

The message backout count survives restarts of the queue manager. However, to ensure that the count is accurate, information has to be *hardened* (recorded on disk or other permanent storage device) each time that an MQGET call retrieves a message within a unit of work for this queue. If this is not done, the queue manager fails, and the MQGET call backs out, the count might or might not be incremented.

Hardening information for each MQGET call within a unit of work, however, imposes additional processing cost, so set the *HardenGetBackout* attribute to MQQA_BACKOUT_HARDENED only if it is essential that the count is accurate.

On IBM i, UNIX systems, and Windows, the message backout count is always hardened, regardless of the setting of this attribute.

The following values are possible:

MQQA_BACKOUT_HARDENED
Hardening is used to ensure that the backout count for messages on this queue is accurate.

MQQA_BACKOUT_NOT_HARDENED
Hardening is not used to ensure that the backout count for messages on this queue is accurate. The count might therefore be lower than it should be.

To determine the value of this attribute, use the MQIA_HARDEN_GET_BACKOUT selector with the MQINQ call.

IndexType (MQLONG):

This specifies the type of index that the queue manager maintains for messages on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

The type of index required depends on how the application retrieves messages, and whether the queue is a shared queue or a nonshared queue (see QSGDisp attribute). The following values are possible for *IndexType*:

MQIT_NONE
No index is maintained by the queue manager for this queue. Use this value for queues that are typically processed sequentially, that is, without using any selection criteria on the MQGET call.

MQIT_MSG_ID
The queue manager maintains an index that uses the message identifiers of the messages on the queue. Use this value queues where the application typically retrieves messages using the message identifier as the selection criterion on the MQGET call.

MQIT_CORREL_ID
The queue manager maintains an index that uses the correlation identifiers of the messages on the queue. Use this value for queues where the application typically retrieves messages using the correlation identifier as the selection criterion on the MQGET call.

MQIT_MSG_TOKEN
The queue manager maintains an index that uses the message tokens of the messages on the queue for use with the workload manager (WLM) functions of z/OS.

You *must* specify this option for WLM-managed queues; do not specify it for any other type of queue. Also, do not use this value for a queue where an application is not using the z/OS workload manager functions, but is retrieving messages using the message token as a selection criterion on the MQGET call.

MQIT_GROUP_ID
The queue manager maintains an index that uses the group identifiers of the messages on the queue. This value *must* be used for queues where the application retrieves messages using the MQGMO_LOGICAL_ORDER option on the MQGET call.

A queue with this index type cannot be a transmission queue. A shared queue with this index type must be defined to map to a CFSTRUCT object at CFLEVEL(3) or CFLEVEL(4).

Note:

1. The physical order of messages on a queue with index type MQIT_GROUP_ID is not defined, as the queue is optimized for efficient retrieval of messages using the MQGMO_LOGICAL_ORDER option on the MQGET call. This means that the physical order of the messages is not typically the order in which the messages arrived on the queue.
2. If an MQIT_GROUP_ID queue has a *MsgDeliverySequence* of MQMDS_PRIORITY, the queue manager uses message priorities 0 and 1 to optimize the retrieval of messages in logical order. As a result, the first message in a group must not have a priority of zero or one; if it does, the message is processed as though it had a priority of two. The *Priority* field in the MQMD structure is not changed.

For more information about message groups, see the description of the group and segment options in MQGMO - Options field.

The index type that should be used in various cases is shown in Table 221 on page 2153 and Table 222 on page 2153.

Table 221. Suggested or required values of queue index type when MQGMO_LOGICAL_ORDER not specified

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	Any	Any
Selection using one identifier:		
Message identifier	MQIT_MSG_ID suggested	MQIT_NONE or MQIT_MSG_ID required; MQIT_MSG_ID suggested
Correlation identifier	MQIT_CORREL_ID suggested	MQIT_CORREL_ID required
Group identifier	MQIT_GROUP_ID suggested	MQIT_GROUP_ID required
Selection using two identifiers:		
Message identifier plus correlation identifier	MQIT_MSG_ID or MQIT_CORREL_ID suggested	MQIT_NONE or MQIT_MSG_ID or MQIT_CORREL_ID required (For efficiency, it is suggested that the index type is chosen to match the MQMD field which will have the most distinct keys)
Message identifier plus group identifier	MQIT_MSG_ID or MQIT_GROUP_ID suggested	Not supported
Correlation identifier plus group identifier	MQIT_CORREL_ID or MQIT_GROUP_ID suggested	Not supported
Selection using three identifiers:		
Message identifier plus correlation identifier plus group identifier	MQIT_MSG_ID or MQIT_CORREL_ID or MQIT_GROUP_ID suggested	Not supported
Selection using group-related criteria:		
Group identifier plus message sequence number	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Message sequence number (must be 1)	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using message token:		
Message token for application use	Do not use MQIT_MSG_TOKEN	
Message token for WLM use	MQIT_MSG_TOKEN required	Not supported

Table 222. Suggested or required values of queue index type when MQGMO_LOGICAL_ORDER specified

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
None	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using one identifier:		
Message identifier	MQIT_GROUP_ID required	Not supported
Correlation identifier	MQIT_GROUP_ID required	Not supported
Group identifier	MQIT_GROUP_ID required	MQIT_GROUP_ID required
Selection using two identifiers:		
Message identifier plus correlation identifier	MQIT_GROUP_ID required	Not supported
Message identifier plus group identifier	MQIT_GROUP_ID required	Not supported
Correlation identifier plus group identifier	MQIT_GROUP_ID required	Not supported

Table 222. Suggested or required values of queue index type when MQGMO_LOGICAL_ORDER specified (continued)

Selection criteria on MQGET call	Index type for nonshared queue	Index type for shared queue
Selection using three identifiers:		
Message identifier plus correlation identifier plus group identifier	MQIT_GROUP_ID required	Not supported

To determine the value of this attribute, use the MQIA_INDEX_TYPE selector with the MQINQ call.

This attribute is supported only on z/OS.

InhibitGet (MQLONG):

This controls whether get operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
X	X	X		

If the queue is an alias queue, get operations must be allowed for both the alias and the base queue at the time of the get operation, for the MQGET call to succeed. The value is one of the following:

MQQA_GET_INHIBITED
Get operations are inhibited.

MQGET calls fail with reason code MQRC_GET_INHIBITED. This includes MQGET calls that specify MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT.

Note: If an MQGET call operating within a unit of work completes successfully, changing the value of the *InhibitGet* attribute subsequently to MQQA_GET_INHIBITED does not prevent the unit of work being committed.

MQQA_GET_ALLOWED
Get operations are allowed.

To determine the value of this attribute, use the MQIA_INHIBIT_GET selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InhibitPut (MQLONG):

This controls whether put operations for this queue are allowed.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

If there is more than one definition in the queue-name resolution path, put operations must be allowed for *every* definition in the path (including any queue-manager alias definitions) at the time of the put operation, for the MQPUT or MQPUT1 call to succeed. The value is one of the following:

MQQA_PUT_INHIBITED
Put operations are inhibited.

MQPUT and MQPUT1 calls fail with reason code MQRC_PUT_INHIBITED.

Note: If an MQPUT call operating within a unit of work completes successfully, changing the value of the *InhibitPut* attribute subsequently to MQQA_PUT_INHIBITED does not prevent the unit of work being committed.

MQQA_PUT_ALLOWED
Put operations are allowed.

To determine the value of this attribute, use the MQIA_INHIBIT_PUT selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

InitiationQName (MQCHAR48):

This is the name of a queue defined on the local queue manager; the queue must be of type MQQT_LOCAL.

Local	Model	Alias	Remote	Cluster
X				

The queue manager sends a trigger message to the initiation queue when application start-up is required as a result of a message arriving on the queue to which this attribute belongs. The initiation queue must be monitored by a trigger monitor application that starts the appropriate application after receipt of the trigger message.

To determine the value of this attribute, use the MQCA_INITIATION_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

MaxMsgLength (MQLONG):

This is an upper limit for the length of the longest *physical* message that can be placed on the queue.

Local	Model	Alias	Remote	Cluster
X	X			

However, because the *MaxMsgLength* queue attribute can be set independently of the *MaxMsgLength* queue-manager attribute, the actual upper limit for the length of the longest physical message that can be placed on the queue is the lesser of those two values.

If the queue manager supports segmentation, it is possible for an application to put a *logical* message that is longer than the lesser of the two *MaxMsgLength* attributes, but only if the application specifies the MQMF_SEGMENTATION_ALLOWED flag in MQMD. If that flag is specified, the upper limit for the length of a logical message is 999 999 999 bytes, but usually resource constraints imposed by the operating system, or by the environment in which the application is running, result in a lower limit.

An attempt to place on the queue a message that is too long fails with one of the following reason codes:

- MQRC_MSG_TOO_BIG_FOR_Q if the message is too big for the queue
- MQRC_MSG_TOO_BIG_FOR_Q_MGR if the message is too big for the queue manager, but not too big for the queue

The lower limit for the *MaxMsgLength* attribute is zero; the upper limit is 100 MB (104 857 600 bytes).

For more information, see MQPUT - BufferLength parameter.

To determine the value of this attribute, use the MQIA_MAX_MSG_LENGTH selector with the MQINQ call.

MaxQDepth (MQLONG):

This is the defined upper limit for the number of physical messages that can exist on the queue at any one time.

Local	Model	Alias	Remote	Cluster
X	X			

An attempt to put a message on a queue that already contains *MaxQDepth* messages fails with reason code MQRC_Q_FULL.

Unit-of-work processing and the segmentation of messages can both cause the actual number of physical messages on the queue to exceed *MaxQDepth*. However, this does not affect the retrievability of the messages; *all* messages on the queue can be retrieved using the MQGET call.

The value of this attribute is zero or greater. The upper limit is determined by the environment:

- On AIX, HP-UX, z/OS, Solaris, Linux, and Windows, the value cannot exceed 999 999 999.
- On IBM i, the value cannot exceed 640 000.

Note: The storage space available to the queue might be exhausted even if there are fewer than *MaxQDepth* messages on the queue.

To determine the value of this attribute, use the MQIA_MAX_Q_DEPTH selector with the MQINQ call.

MsgDeliverySequence (MQLONG):

Local	Model	Alias	Remote	Cluster
X	X			

This determines the order in which the MQGET call returns messages to the application :

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

An MQGET call returns the *first* message that satisfies the selection criteria specified on the call, regardless of the priority of the message.

MQMDS_PRIORITY
Messages are returned in priority order.

An MQGET call returns the *highest-priority* message that satisfies the selection criteria specified on the call. Within each priority level, messages are returned in FIFO order (first in, first out).

- On z/OS, if the queue has an *IndexType* of MQIT_GROUP_ID, the *MsgDeliverySequence* attribute specifies the order in which message groups are returned to the application. The particular sequence in which the groups are returned is determined by the position or priority of the first message in each group. The physical order of messages on the queue is not defined, as the queue is optimized for efficient retrieval of messages using the MQGMO_LOGICAL_ORDER option on the MQGET call.
- On z/OS, if *IndexType* is MQIT_GROUP_ID and *MsgDeliverySequence* is MQMDS_PRIORITY, the queue manager uses message priorities zero and one to optimize the retrieval of messages in logical order. As a result, the first message in a group must not have a priority of zero or one; if it does, the message is processed as though it had a priority of two. The *Priority* field in the MQMD structure is not changed.

If the relevant attributes are changed while there are messages on the queue, the delivery sequence is as follows:

- The order in which messages are returned by the MQGET call is determined by the values of the *MsgDeliverySequence* and *DefPriority* attributes in force for the queue at the time that the message arrives on the queue:
  - If *MsgDeliverySequence* is MQMDS_FIFO when the message arrives, the message is placed on the queue as though its priority were *DefPriority*. This does not affect the value of the *Priority* field in the message descriptor of the message; that field retains the value it had when the message was first put.
  - If *MsgDeliverySequence* is MQMDS_PRIORITY when the message arrives, the message is placed on the queue at the place appropriate to the priority given by the *Priority* field in the message descriptor.

  If the value of the *MsgDeliverySequence* attribute is changed while there are messages on the queue, the order of the messages on the queue is not changed.

  If the value of the *DefPriority* attribute is changed while there are messages on the queue, the messages are not necessarily delivered in FIFO order, even though the *MsgDeliverySequence* attribute is set to MQMDS_FIFO; those that were placed on the queue at the higher priority are delivered first.

To determine the value of this attribute, use the MQIA_MSG_DELIVERY_SEQUENCE selector with the MQINQ call.

NonPersistentMessageClass (MQLONG):

The reliability goal for nonpersistent messages.

Local	Model	Alias	Remote	Cluster
X	X			

This specifies the circumstances under which nonpersistent messages put on this queue are discarded:

MQNPM_CLASS_NORMAL
Nonpersistent messages are limited to the lifetime of the queue manager session; the messages are discarded in the event of a queue manager restart. This is valid only for non-shared queues, and is the default value.

MQNPM_CLASS_HIGH
The queue manager attempts to retain nonpersistent messages for the lifetime of the queue. Nonpersistent messages might still be lost in the event of a failure. This value is enforced for shared queues.

To determine the value of this attribute, use the MQIA_NPM_CLASS selector with the MQINQ call.

OpenInputCount (MQLONG):

This is the number of handles that are currently valid for removing messages from the queue by means of the MQGET call.

Local	Model	Alias	Remote	Cluster
X				

It is the total number of such handles known to the *local* queue manager. If the queue is a shared queue, the count does not include opens for input that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue that resolves to this queue was opened for input. The count does not include handles where the queue was opened for actions that did not include input (for example, a queue opened only for browse).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_OPEN_INPUT_COUNT selector with the MQINQ call.

OpenOutputCount (MQLONG):

This is the number of handles that are currently valid for adding messages to the queue by means of the MQPUT call.

Local	Model	Alias	Remote	Cluster
X				

It is the total number of such handles known to the *local* queue manager; it does not include opens for output that were performed for this queue at remote queue managers. If the queue is a shared queue, the count does not include opens for output that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

The count includes handles where an alias queue that resolves to this queue was opened for output. The count does not include handles where the queue was opened for actions that did not include output (for example, a queue opened only for inquire).

The value of this attribute fluctuates as the queue manager operates.

To determine the value of this attribute, use the MQIA_OPEN_OUTPUT_COUNT selector with the MQINQ call.

ProcessName (MQCHAR48):

This is the name of a process object that is defined on the local queue manager. The process object identifies a program that can service the queue.

Local	Model	Alias	Remote	Cluster
X	X			

To determine the value of this attribute, use the MQCA_PROCESS_NAME selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_NAME_LENGTH.

PropertyControl (MQLONG):

Specifies how message properties are handled for messages that are retrieved from queues using the MQGET call with the MQGMO_PROPERTIES_AS_Q_DEF option.

Local	Model	Alias	Remote	Cluster
X	X	X		

The value is one of the following:

MQPROP_ALL
All properties of the message are included with the message when it is delivered to the application. The properties, except those in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data. If a message handle is supplied then the behavior is to return the properties in the message handle.

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of mcd., jms., usr. or mqext., all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those contained in the message descriptor (or extension), are discarded and are no longer accessible to the application. This is the default value; it allows applications which expect JMS related properties to be in an MQRFH2 header in the message data to continue to work unmodified. If a message handle is supplied then the behavior is to return the properties in the message handle..

MQPROP_FORCE_MQRFH2
Properties are always returned in the message data in an MQRFH2 header regardless of whether the application specifies a message handle. A valid message handle supplied in the MsgHandle field of the MQGMO structure on the MQGET call is ignored. Properties of the message are not accessible via the message handle.

MQPROP_NONE
All properties of the message, except those in the message descriptor (or extension), are removed from the message before the message is delivered to the application. If a message handle is supplied then the behavior is to return the properties in the message handle.

This parameter is applicable to Local, Alias and Model queues. To determine its value, use the MQIA_PROPERTY_CONTROL selector with the MQINQ call.

QDepthHighEvent (MQLONG):

This controls whether Queue Depth High events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold (see the *QDepthHighLimit* attribute).

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_Q_DEPTH_HIGH_EVENT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthHighLimit (MQLONG):

This is the threshold against which the queue depth is compared to generate a Queue Depth High event.

Local	Model	Alias	Remote	Cluster
X	X			

This event indicates that an application has put a message on a queue, and that this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See QDepthHighEvent attribute.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and is greater than or equal to 0 and less than or equal to 100. The default value is 80.

To determine the value of this attribute, use the MQIA_Q_DEPTH_HIGH_LIMIT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthLowEvent (MQLONG):

This controls whether Queue Depth Low events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

A Queue Depth Low event indicates that an application has retrieved a message from a queue, and that this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold (see QDepthLowLimit attribute).

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_EVENT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthLowLimit (MQLONG):

This is the threshold against which the queue depth is compared to generate a Queue Depth Low event.

Local	Model	Alias	Remote	Cluster
X	X			

This event indicates that an application has retrieved a message from a queue, and that this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See QDepthLowEvent attribute.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and is greater than or equal to 0 and less than or equal to 100. The default value is 20.

To determine the value of this attribute, use the MQIA_Q_DEPTH_LOW_LIMIT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDepthMaxEvent (MQLONG):

This controls whether Queue Full events are generated. A Queue Full event indicates that a put to a queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value.

Local	Model	Alias	Remote	Cluster
X	X			

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_Q_DEPTH_MAX_EVENT selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QDesc (MQCHAR64):

Use this field for descriptive commentary.

Local	Model	Alias	Remote	Cluster
X	X	X	X	X

The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_Q_DESC selector with the MQINQ call. The length of this attribute is given by MQ_Q_DESC_LENGTH.

QName (MQCHAR48):

This is the name of a queue defined on the local queue manager.

Local	Model	Alias	Remote	Cluster
X		X	X	X

All queues defined on a queue manager share the same queue namespace. Therefore, an MQQT_LOCAL queue and an MQQT_ALIAS queue cannot have the same name.

To determine the value of this attribute, use the MQCA_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

QServiceInterval (MQLONG):

This is the service interval used for comparison to generate Service Interval High and Service Interval OK events.

Local	Model	Alias	Remote	Cluster
X	X			

See QServiceIntervalEvent attribute.

The value is in units of milliseconds, and is greater than or equal to zero, and less than or equal to 999 999 999.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL selector with the MQINQ call.

This attribute is supported on z/OS, but the MQINQ call cannot be used to determine its value.

QServiceIntervalEvent (MQLONG):

This controls whether Service Interval High or Service Interval OK events are generated.

Local	Model	Alias	Remote	Cluster
X	X			

- A Service Interval High event is generated when a check indicates that no messages have been retrieved from the queue for at least the time indicated by the *QServiceInterval* attribute.
- A Service Interval OK event is generated when a check indicates that messages have been retrieved from the queue within the time indicated by the *QServiceInterval* attribute.

Note: The value of this attribute can change dynamically.

The value is one of the following:

MQQSIE_HIGH
Queue Service Interval High events enabled.
- Queue Service Interval High events are **enabled** and
- Queue Service Interval OK events are **disabled**.

MQQSIE_OK
Queue Service Interval OK events enabled.
- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

MQQSIE_NONE
No queue service interval events enabled.
- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

For shared queues, the value of this attribute is ignored; the value MQQSIE_NONE is assumed.

For more information about events, see Event monitoring.

To determine the value of this attribute, use the MQIA_Q_SERVICE_INTERVAL_EVENT selector with the MQINQ call.

On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

QSGDisp (MQLONG):

This specifies the disposition of the queue.

Local	Model	Alias	Remote	Cluster
X		X	X	

The value is one of the following:

MQQSGD_Q_MGR
The object has queue-manager disposition. This means that the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY
The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but by using MQSC commands, you can alter each copy so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

MQQSGD_SHARED
The object has shared disposition. This means that there exists in the shared repository a single instance of the object that is known to all queue managers in the queue-sharing group. When a queue manager in the group accesses the object, it accesses the single shared instance of the object.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

QueueAccounting (MQLONG):

Local	Model	Alias	Remote	Cluster
X	X	X	X	

This controls the collection of accounting data for the queue. For accounting data to be collected for this queue, accounting data for this connection must also be enabled, using either the QMGR attribute ACCTQ or the Options field in the MQCNO structure on the MQCONNX call.

This attribute has one of the following values:

MQMON_Q_MGR
Accounting data for this queue is collected based on the setting of the QMGR attribute ACCTQ. This is the default setting.

MQMON_OFF
Do not collect accounting data for this queue.

MQMON_ON
Collect accounting data for this queue.

To determine the value of this attribute, use the MQIA_ACCOUNTING_Q selector with the MQINQ call.

QueueMonitoring (MQLONG):

Controls the collection of online monitoring data for queues.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQMON_Q_MGR
Collect monitoring data according to the setting of the *QueueMonitoring* queue manager attribute. This is the default value.

MQMON_OFF
Online monitoring data collection is turned off for this queue.

MQMON_LOW
If the value of the *QueueMonitoring* queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a low rate of data collection for this queue.

MQMON_MEDIUM
If the value of the *QueueMonitoring* queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a moderate rate of data collection for this queue.

MQMON_HIGH
If the value of the *QueueMonitoring* queue manager attribute is not MQMON_NONE, online monitoring data collection is turned on, with a high rate of data collection for this queue.

To determine the value of this attribute, use the MQIA_MONITORING_Q selector with the MQINQ call.

QueueStatistics (MQCHAR12):

Local	Model	Alias	Remote	Cluster
X	X	X	X	

This controls the collection of statistics data for the queue.

This attribute has one of the following values:

MQMON_Q_MGR
Accounting data for this queue is collected based on the setting of the QMGR attribute STATQ. This is the default setting.

MQMON_OFF
Switch off statistics data collection for this queue.

MQMON_ON
Switch on statistics data collection for this queue.

QType (MQLONG):

Local	Model	Alias	Remote	Cluster
X		X	X	X

This is the type of queue; it has one of the following values:

MQQT_ALIAS
Alias queue definition.

MQQT_CLUSTER
Cluster queue.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

To determine the value of this attribute, use the MQIA_Q_TYPE selector with the MQINQ call.

RemoteQMgrName (MQCHAR48):

Local	Model	Alias	Remote	Cluster
			X	

This is the name of the remote queue manager on which the queue *RemoteQName* is defined. If the *RemoteQName* queue has a *QSGDisp* value of MQQSGD_COPY or MQQSGD_SHARED, *RemoteQMgrName* can be the name of the queue-sharing group that owns *RemoteQName*.

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank and must not be the name of the local queue manager. If *XmitQName* is blank, the local queue with the same name as *RemoteQMgrName* is used as the transmission queue. If there is no queue with the name *RemoteQMgrName*, the queue identified by the *DefXmitQName* queue-manager attribute is used.

If this definition is used for a queue-manager alias, *RemoteQMgrName* is the name of the queue manager that is being aliased. It can be the name of the local queue manager. Otherwise, if *XmitQName* is blank when the open occurs, there must be a local queue with a name that is the same as *RemoteQMgrName*; this queue is used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager that is to be the *ReplyToQMgr*.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the MQCA_REMOTE_Q_MGR_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_MGR_NAME_LENGTH.

RemoteQName (MQCHAR48):

Local	Model	Alias	Remote	Cluster
			X	

This is the name of the queue as it is known on the remote queue manager *RemoteQMgrName*.

If an application opens the local definition of a remote queue, when the open occurs *RemoteQName* must not be blank.

If this definition is used for a queue-manager alias definition, when the open occurs *RemoteQName* must be blank.

If the definition is used for a reply-to alias, this name is the name of the queue that is to be the *ReplyToQ*.

Note: No validation is performed on the value specified for this attribute when the queue definition is created or modified.

To determine the value of this attribute, use the MQCA_REMOTE_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

RetentionInterval (MQLONG):

This is the period of time for which to retain the queue. After this time has elapsed, the queue is eligible for deletion.

Local	Model	Alias	Remote	Cluster
X	X			

The time is measured in hours, counting from the date and time when the queue was created. The creation date and time of the queue are recorded in the *CreationDate* and *CreationTime* attributes.

This information is provided to enable a housekeeping application or the operator to identify and delete queues that are no longer required.

Note: The queue manager never takes any action to delete queues based on this attribute, or to prevent the deletion of queues with a retention interval that has not expired; it is the user's responsibility to take any required action.

Use a realistic retention interval to prevent the accumulation of permanent dynamic queues (see DefinitionType attribute). However, this attribute can also be used with predefined queues.

To determine the value of this attribute, use the MQIA_RETENTION_INTERVAL selector with the MQINQ call.

Scope (MQLONG):

This controls whether an entry for this queue also exists in a cell directory.

Local	Model	Alias	Remote	Cluster
X		X	X	

A cell directory is provided by an installable Name service. The value is one of the following:

MQSCO_Q_MGR
The queue definition has queue-manager scope: the definition of the queue does not extend beyond the queue manager that owns it. To open the queue for output from some other queue manager, either the name of the owning queue manager must be specified, or the other queue manager must have a local definition of the queue.

MQSCO_CELL
The queue definition has cell scope: the queue definition is also placed in a cell directory available to all the queue managers in the cell. The queue can be opened for output from any of the queue managers in the cell by specifying the name of the queue; the name of the queue manager that owns the queue need not be specified. However, the queue definition is not available to any queue manager in the cell that also has a local definition of a queue with that name, as the local definition takes precedence.

A cell directory is provided by an installable Name service.

Model and dynamic queues cannot have cell scope.

This value is only valid if a name service supporting a cell directory has been configured.

To determine the value of this attribute, use the MQIA_SCOPE selector with the MQINQ call.

Support for this attribute is subject to the following restrictions:
- On IBM i, the attribute is supported, but only MQSCO_Q_MGR is valid.
- On z/OS, the attribute is not supported.

Shareability (MQLONG):

This indicates whether the queue can be opened for input multiple times concurrently.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQQA_SHAREABLE
Queue is shareable.

Multiple opens with the MQOO_INPUT_SHARED option are allowed.

MQQA_NOT_SHAREABLE
Queue is not shareable.

An MQOPEN call with the MQOO_INPUT_SHARED option is treated as MQOO_INPUT_EXCLUSIVE.

To determine the value of this attribute, use the MQIA_SHAREABILITY selector with the MQINQ call.

StorageClass (MQCHAR8):

This is a user-defined name that defines the physical storage used to hold the queue. In practice, a message is written to disk only if it needs to be paged out of its memory buffer.

Local	Model	Alias	Remote	Cluster
X	X			

To determine the value of this attribute, use the MQCA_STORAGE_CLASS selector with the MQINQ call. The length of this attribute is given by MQ_STORAGE_CLASS_LENGTH.

This attribute is supported only on z/OS.

TriggerControl (MQLONG):

This controls whether trigger messages are written to an initiation queue to start an application to service the queue.

Local	Model	Alias	Remote	Cluster
X	X			

This is one of the following:

MQTC_OFF
No trigger messages are to be written for this queue. The value of *TriggerType* is irrelevant in this case.

MQTC_ON
Trigger messages are to be written for this queue when the appropriate trigger events occur.

To determine the value of this attribute, use the MQIA_TRIGGER_CONTROL selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerData (MQCHAR64):

This is free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue.

Local	Model	Alias	Remote	Cluster
X	X			

The content of this data is of no significance to the queue manager. It is meaningful either to the trigger-monitor application that processes the initiation queue, or to the application that the trigger monitor starts.

The character string must not contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_TRIGGER_DATA selector with the MQINQ call. To change the value of this attribute, use the MQSET call. The length of this attribute is given by MQ_TRIGGER_DATA_LENGTH.

TriggerDepth (MQLONG):

Local	Model	Alias	Remote	Cluster
X	X			

This is the number of messages of priority *TriggerMsgPriority* or greater that must be on the queue before a trigger message is written. This applies when *TriggerType* is set to MQTT_DEPTH. The value of *TriggerDepth* is one or greater. This attribute is not used otherwise.

To determine the value of this attribute, use the MQIA_TRIGGER_DEPTH selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerMsgPriority (MQLONG):

This is the message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when deciding whether to generate a trigger message).

Local	Model	Alias	Remote	Cluster
X	X			

TriggerMsgPriority can be in the range zero (lowest) through *MaxPriority* (highest; see MaxPriority attribute); a value of zero causes all messages to contribute to the generation of trigger messages.

To determine the value of this attribute, use the MQIA_TRIGGER_MSG_PRIORITY selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

TriggerType (MQLONG):

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue.

Local	Model	Alias	Remote	Cluster
X	X			

It has one of the following values:

MQTT_NONE
No trigger messages are written as a result of messages on this queue. This has the same effect as setting *TriggerControl* to MQTC_OFF.

MQTT_FIRST
A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue changes from 0 to 1.

MQTT_EVERY
A trigger message is written whenever a message of priority *TriggerMsgPriority* or greater arrives on the queue.

MQTT_DEPTH
A trigger message is written whenever the number of messages of priority *TriggerMsgPriority* or greater on the queue equals or exceeds *TriggerDepth*. After the trigger message has been written, *TriggerControl* is set to MQTC_OFF to prevent further triggering until it is explicitly turned on again.

To determine the value of this attribute, use the MQIA_TRIGGER_TYPE selector with the MQINQ call. To change the value of this attribute, use the MQSET call.

Usage (MQLONG):

This indicates what the queue is used for.

Local	Model	Alias	Remote	Cluster
X	X			

The value is one of the following:

MQUS_NORMAL
This is a queue that applications use when putting and getting messages; the queue is not a transmission queue.

MQUS_TRANSMISSION
This is a queue used to hold messages destined for remote queue managers. When an application sends a message to a remote queue, the local queue manager stores the message temporarily on the appropriate transmission queue in a special format. A message channel agent then reads the message from the transmission queue, and transports the message to the remote queue manager. For more information about transmission queues, see Defining a transmission queue.

Only privileged applications can open a transmission queue for MQOO_OUTPUT to put messages on it directly. Usually, only utility applications do this. Ensure that the message data format is correct (see "MQXQH - Transmission-queue header" on page 1918) or errors might occur during the transmission process. Context is not passed or set unless one of the MQPMO_*_CONTEXT context options is specified.

To determine the value of this attribute, use the MQIA_USAGE selector with the MQINQ call.

XmitQName (MQCHAR48):

This is the transmission queue name. If this attribute is nonblank when an open occurs, either for a remote queue or for a queue-manager alias definition, it specifies the name of the local transmission queue to be used for forwarding the message.

Local	Model	Alias	Remote	Cluster
			X	

If *XmitQName* is blank, the local queue with a name that is the same as *RemoteQMgrName* is used as the transmission queue. If there is no queue with the name *RemoteQMgrName*, the queue identified by the *DefXmitQName* queue-manager attribute is used.

This attribute is ignored if the definition is being used as a queue-manager alias and *RemoteQMgrName* is the name of the local queue manager. It is also ignored if the definition is used as a reply-to queue alias definition.

To determine the value of this attribute, use the MQCA_XMIT_Q_NAME selector with the MQINQ call. The length of this attribute is given by MQ_Q_NAME_LENGTH.

Attributes for namelists:

The following table summarizes the attributes that are specific to namelists. The attributes are described in alphabetical order.

Namelists are supported on all WebSphere MQ systems, plus WebSphere MQ MQI clients connected to these systems.

Note: The names of the attributes shown in this section are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) Commands for more information.

Table 223. Attributes for namelists

Attribute	Description
AlterationDate	Date when definition was last changed
AlterationTime	Time when definition was last changed
NameCount	Number of names in namelist
NamelistDesc	Namelist description
NamelistName	Namelist name
Names	A list of *NameCount* names
NamelistType	Namelist type
QSGDisp	Queue-sharing group disposition

AlterationDate (MQCHAR12):

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8):

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

NameCount (MQLONG):

This is the number of names in the namelist. It is greater than or equal to zero. The following value is defined:

MQNC_MAX_NAMELIST_NAME_COUNT
Maximum number of names in a namelist.

To determine the value of this attribute, use the MQIA_NAME_COUNT selector with the MQINQ call.

NamelistDesc (MQCHAR64):

Use this field for descriptive commentary; its value is established by the definition process. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, this field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_NAMELIST_DESC selector with the MQINQ call.

The length of this attribute is given by MQ_NAMELIST_DESC_LENGTH.

NamelistName (MQCHAR48):

This is the name of a namelist that is defined on the local queue manager. For more information about namelist names, see the Other object names section.

Each namelist has a name that is different from the names of other namelists belonging to the queue manager, but might duplicate the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the MQCA_NAMELIST_NAME selector with the MQINQ call.

The length of this attribute is given by MQ_NAMELIST_NAME_LENGTH.

NamelistType (MQLONG):

This specifies the nature of the names in the namelist, and indicates how the namelist is used. It is one of the following values:

MQNT_NONE
Namelist with no assigned type.

MQNT_Q
Namelist containing the names of queues.

MQNT_CLUSTER
Namelist containing the names of clusters.

MQNT_AUTH_INFO
Namelist containing the names of authentication-information objects.

To determine the value of this attribute, use the MQIA_NAMELIST_TYPE selector with the MQINQ call.

This attribute is supported only on z/OS.

Names (MQCHAR48xNameCount):

This is a list of *NameCount* names, where each name is the name of an object that is defined to the local queue manager. For more information about object names, see Rules for naming IBM WebSphere MQ objects.

To determine the value of this attribute, use the MQCA_NAMES selector with the MQINQ call.

The length of each name in the list is given by MQ_OBJECT_NAME_LENGTH.

QSGDisp (MQLONG):

This specifies the disposition of the namelist. The value is one of the following:

MQQSGD_Q_MGR
The object has queue-manager disposition: the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY
The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all

copies have the same attributes, but you can alter each copy, using MQSC commands, so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

Attributes for process definitions:

The following table summarizes the attributes that are specific to process definitions. The attributes are described in alphabetical order.

Note: The names of the attributes in this section are descriptive names used with the MQINQ and MQSET calls; the names are the same as for the PCF commands. When MQSC commands are used to define, alter, or display attributes, alternative short names are used; see Script (MQSC) Commands for more information.

Table 224. Attributes for process definitions

Attribute	Description
AlterationDate	Date when definition was last changed
AlterationTime	Time when definition was last changed
ApplId	Application identifier
ApplType	Application type
EnvData	Environment data
ProcessDesc	Process description
ProcessName	Process name
QSGDisp	Queue-sharing group disposition
UserData	User data

AlterationDate (MQCHAR12):

This is the date when the definition was last changed. The format of the date is `YYYY-MM-DD`, padded with two trailing blanks to make the length 12 bytes.

To determine the value of this attribute, use the MQCA_ALTERATION_DATE selector with the MQINQ call. The length of this attribute is given by MQ_DATE_LENGTH.

AlterationTime (MQCHAR8):

This is the time when the definition was last changed. The format of the time is `HH.MM.SS`.

To determine the value of this attribute, use the MQCA_ALTERATION_TIME selector with the MQINQ call. The length of this attribute is given by MQ_TIME_LENGTH.

ApplId (MQCHAR256):

This is a character string that identifies the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *ApplId* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ requires *ApplId* to be the name of an executable program. The following notes apply to the environments indicated:

- On z/OS, *ApplId* must be:

  - A CICS transaction identifier, for applications started using the CICS trigger-monitor transaction CKTI
  - An IMS transaction identifier, for applications started using the IMS trigger monitor CSQQTRMN
- On Windows systems, the program name can be prefixed with a drive and directory path.
- On UNIX systems, the program name can be prefixed with a directory path.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_APPL_ID selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_APPL_ID_LENGTH.

ApplType (MQLONG):

This identifies the nature of the program to be started in response to the receipt of a trigger message. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

ApplType can have any value, but the following values are recommended for standard types; restrict user-defined application types to values in the range MQAT_USER_FIRST through MQAT_USER_LAST:

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_BATCH
Batch application

MQAT_BROKER
Broker application

MQAT_CICS
CICS transaction.

MQAT_CICS_BRIDGE
CICS bridge application.

MQAT_CICS_VSE
CICS/VSE transaction.

MQAT_DOS
WebSphere MQ MQI client application on PC DOS.

MQAT_IMS
IMS application.

MQAT_IMS_BRIDGE
IMS bridge application.

MQAT_JAVA
Java application.

MQAT_MVS
MVS or TSO application (same value as MQAT_ZOS).

MQAT_NOTES_AGENT
Lotus Notes Agent application.

MQAT_NSK
HP Integrity NonStop Server application.

MQAT_OS390
OS/390 application (same value as MQAT_ZOS).

MQAT_OS400
IBM i application.

MQAT_RRS_BATCH
RRS batch application.

MQAT_UNIX
UNIX application.

MQAT_UNKNOWN
Application of unknown type.

MQAT_USER
User application.

MQAT_VOS
Stratus VOS application.

MQAT_WINDOWS
16-bit Windows application.

MQAT_WINDOWS_NT
32-bit Windows application.

MQAT_WLM
z/OS workload manager application.

MQAT_XCF
XCF.

MQAT_ZOS
z/OS application.

MQAT_USER_FIRST
Lowest value for user-defined application type.

MQAT_USER_LAST
Highest value for user-defined application type.

To determine the value of this attribute, use the MQIA_APPL_TYPE selector with the MQINQ call.

EnvData (MQCHAR128):

This is a character string that contains environment-related information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

The meaning of *EnvData* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ appends *EnvData* to the parameter list passed to the started application. The parameter list consists of the MQTMC2 structure, followed by one blank, followed by *EnvData* with trailing blanks removed. The following notes apply to the environments indicated:

- On z/OS:
  - *EnvData* is not used by the trigger-monitor applications provided by WebSphere MQ.
  - If ApplType is MQAT_WLM, you can supply default values in EnvData for the ServiceName and ServiceStep fields in the work information header (MQWIH).
- On UNIX systems, *EnvData* can be set to the & character to run the started application in the background.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary.

To determine the value of this attribute, use the MQCA_ENV_DATA selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc (MQCHAR64):

Use this field for descriptive commentary. The content of the field is of no significance to the queue manager, but the queue manager might require that the field contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the queue manager's character set (as defined by the *CodedCharSetId* queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

To determine the value of this attribute, use the MQCA_PROCESS_DESC selector with the MQINQ call.

The length of this attribute is given by MQ_PROCESS_DESC_LENGTH.

ProcessName (MQCHAR48):

This is the name of a process definition that is defined on the local queue manager.

Each process definition has a name that is different from the names of other process definitions belonging to the queue manager. But the name of the process definition might be the same as the names of other queue manager objects of different types (for example, queues).

To determine the value of this attribute, use the MQCA_PROCESS_NAME selector with the MQINQ call.

The length of this attribute is given by MQ_PROCESS_NAME_LENGTH.

QSGDisp (MQLONG):

This specifies the disposition of the process definition. The value is one of the following:

MQQSGD_Q_MGR
: The object has queue-manager disposition: the object definition is known only to the local queue manager; the definition is not known to other queue managers in the queue-sharing group.

  Each queue manager in the queue-sharing group can have an object with the same name and type as the current object, but these are separate objects and there is no correlation between them. Their attributes are not constrained to be the same as each other.

MQQSGD_COPY
: The object is a local copy of a master object definition that exists in the shared repository. Each queue manager in the queue-sharing group can have its own copy of the object. Initially, all copies have the same attributes, but you can alter each copy, using MQSC commands, so that its attributes differ from those of the other copies. The attributes of the copies are resynchronized when the master definition in the shared repository is altered.

To determine the value of this attribute, use the MQIA_QSG_DISP selector with the MQINQ call.

This attribute is supported only on z/OS.

UserData (MQCHAR128):

`UserData` is a character string that contains user information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue, or the application that is started by the trigger monitor. The information is sent to the initiation queue as part of the trigger message.

The meaning of *UserData* is determined by the trigger-monitor application. The trigger monitor provided by WebSphere MQ passes *UserData* to the started application as part of the parameter list. The parameter list consists of the MQTMC2 structure (containing *UserData*), followed by one blank, followed by *EnvData* with trailing blanks removed.

The character string cannot contain any nulls. It is padded to the right with blanks if necessary. For Microsoft Windows, the character string must not contain double quotation marks if the process definition is going to be passed to **runmqtrm**.

To determine the value of this attribute, use the MQCA_USER_DATA selector with the MQINQ call. The length of this attribute is given by MQ_PROCESS_USER_DATA_LENGTH.

Return codes

For each WebSphere MQ Message Queue Interface (MQI) and WebSphere MQ Administration Interface (MQAI) call, a **completion** code and a **reason** code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

Applications must not depend upon errors being checked for in a specific order, except where specifically noted. If more than one completion code or reason code could arise from a call, the particular error reported depends on the implementation.

Applications checking for successful completion following a WebSphere MQ API call must always check the completion code. Do not assume the completion code value, based on the value of the reason code.

Completion codes

The completion code parameter (*CompCode*) allows the caller to see quickly whether the call completed successfully, completed partially, or failed. The following is a list of completion codes, with more detail than is given in the call descriptions:

MQCC_OK
The call completed fully; all output parameters have been set. The *Reason* parameter always has the value MQRC_NONE in this case.

MQCC_WARNING
The call completed partially. Some output parameters might have been set in addition to the *CompCode* and *Reason* output parameters. The *Reason* parameter gives additional information about the partial completion.

MQCC_FAILED
The processing of the call did not complete. The state of the queue manager is unchanged, except where specifically noted. The *CompCode* and *Reason* output parameters have been set; other parameters are unchanged, except where noted.

The reason might be a fault in the application program, or it might be the result of some situation external to the program, for example the user's authority might have been revoked. The *Reason* parameter gives additional information about the error.

Reason codes

The reason code parameter (*Reason*) qualifies the completion code parameter (*CompCode*).

If there is no special reason to report, MQRC_NONE is returned. A successful call returns MQCC_OK and MQRC_NONE.

If the completion code is either MQCC_WARNING or MQCC_FAILED, the queue manager always reports a qualifying reason; details are given under each call description.

Where user exit routines set completion codes and reasons, they must adhere to these rules. In addition, any special reason values defined by user exits must be less than zero, to ensure that they do not conflict with values defined by the queue manager. Exits can set reasons already defined by the queue manager, where appropriate.

Reason codes also occur in:

- The *Reason* field of the MQDLH structure
- The *Feedback* field of the MQMD structure

For complete descriptions of reason codes, see Reason codes.

Rules for validating MQI options

This section lists the situations that produce an MQRC_OPTIONS_ERROR reason code from an MQOPEN, MQPUT, MQPUT1, MQGET, MQCLOSE, or MQSUB call.

MQOPEN call

For the options of the MQOPEN call:

- At least *one* of the following must be specified:
  - MQOO_BROWSE
  - MQOO_INPUT_EXCLUSIVE[1]
  - MQOO_INPUT_SHARED[1]
  - MQOO_INPUT_AS_Q_DEF[1]
  - MQOO_INQUIRE
  - MQOO_OUTPUT
  - MQOO_SET
  - MQOO_BIND_ON_OPEN[2]
  - MQOO_BIND_NOT_FIXED[2]
  - MQOO_BIND_ON_GROUP[2]
  - MQOO_BIND_AS_Q_DEF[2]
- Only *one* of the following is allowed:
  - MQOO_READ_AHEAD
  - MQOO_NO_READ_AHEAD
  - MQOO_READ_AHEAD_AS_Q_DEF

1. Only *one* of the following is allowed:
   - MQOO_INPUT_EXCLUSIVE
   - MQOO_INPUT_SHARED
   - MQOO_INPUT_AS_Q_DEF
2. Only *one* of the following is allowed:
   - MQOO_BIND_ON_OPEN
   - MQOO_BIND_NOT_FIXED
   - MQOO_BIND_ON_GROUP
   - MQOO_BIND_AS_Q_DEF

Note: The options listed above are mutually exclusive. However, as the value of MQOO_BIND_AS_Q_DEF is zero, specifying it with either of the other two bind options does not result in reason code MQRC_OPTIONS_ERROR. MQOO_BIND_AS_Q_DEF is provided to aid program documentation.

- If MQOO_SAVE_ALL_CONTEXT is specified, one of the MQOO_INPUT_* options must also be specified.
- If one of the MQOO_SET_*_CONTEXT or MQOO_PASS_*_CONTEXT options are specified, MQOO_OUTPUT must also be specified.
- If MQOO_CO_OP is specified, MQOO_BROWSE must also be specified
- If MQOO_NO_MULTICAST is specified, MQOO_OUTPUT must also be specified.

MQPUT call

For the put-message options:

- The combination of MQPMO_SYNCPOINT and MQPMO_NO_SYNCPOINT is not allowed.
- Only *one* of the following is allowed:
  - MQPMO_DEFAULT_CONTEXT
  - MQPMO_NO_CONTEXT
  - MQPMO_PASS_ALL_CONTEXT
  - MQPMO_PASS_IDENTITY_CONTEXT
  - MQPMO_SET_ALL_CONTEXT
  - MQPMO_SET_IDENTITY_CONTEXT
- Only *one* of the following is allowed:
  - MQPMO_ASYNC_RESPONSE
  - MQPMO_SYNC_RESPONSE
  - MQPMO_RESPONSE_AS_TOPIC_DEF
  - MQPMO_RESPONSE_AS_Q_DEF
- MQPMO_ALTERNATE_USER_AUTHORITY is not allowed (it is valid only on the MQPUT1 call).

MQPUT1 call

For the put-message options, the rules are the same as for the MQPUT call, except for the following:

- MQPMO_ALTERNATE_USER_AUTHORITY is allowed.
- MQPMO_LOGICAL_ORDER is *not* allowed.

MQGET call

For the get-message options:

- Only *one* of the following is allowed:
  - MQGMO_NO_SYNCPOINT
  - MQGMO_SYNCPOINT
  - MQGMO_SYNCPOINT_IF_PERSISTENT
- Only *one* of the following is allowed:
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_BROWSE_NEXT
  - MQGMO_MSG_UNDER_CURSOR
- MQGMO_SYNCPOINT is not allowed with any of the following:

  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_BROWSE_NEXT
  - MQGMO_LOCK
  - MQGMO_UNLOCK
- MQGMO_SYNCPOINT_IF_PERSISTENT is not allowed with any of the following:
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_BROWSE_NEXT
  - MQGMO_COMPLETE_MSG
  - MQGMO_UNLOCK
- MQGMO_MARK_SKIP_BACKOUT requires MQGMO_SYNCPOINT to be specified.
- The combination of MQGMO_WAIT and MQGMO_SET_SIGNAL is not allowed.
- If MQGMO_LOCK is specified, one of the following must also be specified:
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_BROWSE_NEXT
- If MQGMO_UNLOCK is specified, only the following are allowed:
  - MQGMO_NO_SYNCPOINT
  - MQGMO_NO_WAIT

MQCLOSE call

For the options of the MQCLOSE call:

- The combination of MQCO_DELETE and MQCO_DELETE_PURGE is not allowed.
- Only one of the following is allowed:
  - MQCO_KEEP_SUB
  - MQCO_REMOVE_SUB

MQSUB call

For the options of the MQSUB call:

- At least one of the following must be specified:
  - MQSO_ALTER
  - MQSO_RESUME
  - MQSO_CREATE
- Only one of the following is allowed:
  - MQSO_DURABLE
  - MQSO_NON_DURABLE

  Note: The options listed above are mutually exclusive. However, as the value of MQSO_NON_DURABLE is zero, specifying it with MQSO_DURABLE does not result in reason code MQRC_OPTIONS_ERROR. MQSO_NON_DURABLE is provided to aid program documentation.
- The combination of MQSO_GROUP_SUB and MQSO_MANAGED is not allowed.
- MQSO_GROUP_SUB requires MQSO_SET_CORREL_ID to be specified.
- Only one of the following is allowed:
  - MQSO_ANY_USERID

  - MQSO_FIXED_USERID
- MQSO_NEW_PUBLICATIONS_ONLY is only allowed in combination with MQSO_CREATE.
- The combination of MQSO_PUBLICATIONS_ON_REQUEST and SubLevel greater than 1 is not allowed.
- Only one of the following is allowed:
  - MQSO_WILDCARD_CHAR
  - MQSO_WILDCARD_TOPIC
- MQSO_NO_MULTICAST requires MQSO_MANAGED to be specified.

Queued publish/subscribe command messages

An application can use MQRFH2 command messages to control a queued publish/subscribe application.

An application that is using MQRFH2 for publish/subscribe can send the following command messages to the SYSTEM.BROKER.CONTROL.QUEUE:

- "Delete Publication message"
- "Deregister Subscriber message" on page 2183
- "Publish message" on page 2187
- "Register Subscriber message" on page 2190
- "Request Update message" on page 2194

If you are writing queued publish/subscribe applications, you must understand these messages, the queue manager response message, and the message descriptor (MQMD); see the following information:

- "Queue Manager Response message" on page 2196
- "MQMD settings for publications forwarded by a queue manager" on page 2202
- "MQMD settings in queue manager response messages" on page 2203
- "Publish/subscribe reason codes" on page 2198

The commands are contained in a <psc> folder in the **NameValueData** field of the MQRFH2 header. The message that can be sent by a broker in response to a command message is contained in a <pscr> folder.

The descriptions of each command list the properties that can be contained in a folder. Unless otherwise specified, the properties are optional and can occur only once.

Names of properties are shown as <Command>.

Values must be in string format, for example: Publish.

A string constant representing the value of a property is shown in parentheses, for example: (MQPSC_PUBLISH).

String constants are defined in the header file cmqpsc.h which is supplied with the queue manager.

Delete Publication message:

The **Delete Publication** command message is sent to a queue manager from a publisher, or from another queue manager, to tell the queue manager to delete any retained publications for the specified topics.

This message is sent to a queue monitored by the queue manager's queued publish/subscribe interface.

The input queue should be the queue that the original publication was sent to.

If you have the authority for some, but not all, of the topics that are specified in the **Delete Publication** command message, only those topics are deleted. A **Broker Response** message indicates which topics are not deleted.

Similarly, if a **Publish** command contains more than one topic, a **Delete Publication** command matching some, but not all, of those topics deletes only the publications for the topics that are specified in the **Delete Publication** command.

See "MQMD settings for publications forwarded by a queue manager" on page 2202 for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

Properties:

<Command> (*MQPSC_COMMAND*)
The value is `DeletePub`(*MQPSC_DELETE_PUBLICATION*).

This property must be specified.

<Topic> (*MQPSC_TOPIC*)
The value is a string that contains a topic for which retained publications are to be deleted. Wildcard characters can be included in the string to delete publications on more than one topic.

This property must be specified; it can be repeated for as many topics as needed.

<DelOpt> (*MQPSC_DELETE_OPTION*)
The delete options property can take one of the following values:
Local (*MQPSC_LOCAL*)

All retained publications for the specified topics are deleted at the local queue manager (that is, the queue manager to which this message is sent), whether they were published with the `Local` option or not.

Publications at other queue managers are not affected.

None (*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the `DelOpt` property. If other options are specified at the same time, `None` is ignored.

The default if this property is omitted is that all retained publications for the specified topics are deleted at all queue managers in the network, regardless of whether they were published with the `Local` option.

Example:
Here is an example of `NameValueData` for a **Delete Publication** command message. This is used by the sample application to delete, at the local queue manager, the retained publication that contains the latest score in the match between Team1 and Team2.

```
<psc>
 <Command>DeletePub</Command>
 <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
 <DelOpt>Local</DelOpt>
</psc>
```

Deregister Subscriber message:

The **Deregister Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it no longer wants to receive messages matching the given parameters.

This message is sent to SYSTEM.BROKER.CONTROL.QUEUE, the queue manager's control queue. The user must have the necessary authority to put a message onto this queue.

See MQMD settings for publications forwarded by a queue manager for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

An individual subscription can be deregistered by specifying the corresponding topic, subscription point and filter values of the original subscription. If any of the values were not specified (that is, they took the default values) in the original subscription, they should be omitted when the subscription is deregistered.

All subscriptions for a subscriber, or a group of subscribers, can be deregistered by using the `DeregAll` option. For example, if `DeregAll` is specified, together with a subscription point (but no topic or filter), then all subscriptions for the subscriber on the specified subscription point are deregistered, regardless of the topic and filter. Any combination of topic, filter and subscription point is allowed; if all three are specified only one subscription can match, and the `DeregAll` option is ignored.

The message must be sent by the subscriber that registered the subscription; this is confirmed by checking the subscriber's user ID.

Subscriptions can also be deregistered by a system administrator using MQSC or PCF commands. However, the subscriptions registered with a temporary dynamic queue are associated with the queue, not just the queue name. If the queue is deleted, either explicitly, or by the application disconnecting from the queue manager, it is no longer possible to use the **Deregister Subscriber** command to deregister the subscriptions for that queue. The subscriptions can be deregistered using the developer workbench, and they are removed automatically by the queue manager the next time that it matches a publication to the subscription, or the next time the queue manager restarts. Under normal circumstances, applications should deregister their subscriptions before deleting the queue, or disconnecting from the queue manager.

If a subscriber sends a message to deregister a subscription, and receives a response message to say that this was processed successfully, some publications might still reach the subscriber queue if they were being processed by the queue manager at the same time as the subscription was being deregistered. If the messages are not removed from the queue, there might be a buildup of unprocessed messages on the subscriber queue. If the application executes a loop that includes an MQGET call with the appropriate `CorrelId` after sleeping for a while, these messages are cleared off the queue.

Similarly, if the subscriber uses a permanent dynamic queue, and deregisters and closes the queue with the *MQCO_DELETE_PURGE* option on an MQCLOSE call, the queue might not be empty. If any publications from the queue manager are not yet committed when the queue is deleted, an MQRC_Q_NOT_EMPTY return code is issued by the MQCLOSE call. The application can avoid this problem by sleeping and reissuing the MQCLOSE call from time to time.

Properties:

<Command> (*MQPSC_COMMAND*)
The value is `DeregSub` (*MQPSC_DEREGISTER_SUBSCRIBER*).

This property must be specified.

<Topic> (*MQPSC_TOPIC*)
The value is a string that contains the topic to be deregistered.

This property can, optionally, be repeated if multiple topics are to be deregistered. It can be omitted if `DeregAll` is specified in `<RegOpt>`.

The topics that are specified can be a subset of those that are registered if the subscriber wants to retain subscriptions for other topics. Wildcard characters are allowed, but a topic string that contains wildcard characters must exactly match the corresponding string that was specified in the **Deregister Subscriber** command message.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT*)
The value is a string that specifies the subscription point from which the subscription is to be detached.

This property must not be repeated. It can be omitted if a `<Topic>` is specified, or if `DeregAll` is specified in `<RegOpt>`. If you omit this property, the following happens:

- If you do **not** specify DeregAll, subscriptions matching the <Topic> property (and the <Filter> property, if present) are deregistered from the default subscription point.
- If you specify DeregAll, all subscriptions (matching the <Topic> and <Filter> properties if present) are deregistered from all subscription points.

Note that you cannot specify the default subscription point explicitly. Therefore, there is no way of deregistering all subscriptions from this subscription point only; you must specify the topics.

<SubIdentity> (*MQPSC_SUBSCRIPTION_IDENTITY*)
This is a variable-length string with a maximum length of 64 characters. It is used to represent an application with an interest in a subscription. The queue manager maintains a set of subscriber identities for each subscription. Each subscription can allow its identity set to hold only a single identity, or an unlimited number of identities.

If the SubIdentity is in the identity set for the subscription then it is removed from the set. If the identity set becomes empty as a result of this, the subscription is removed from the queue manager, unless LeaveOnly is specified as a value of the RegOpt property. If the identity set still contains other identities then the subscription is not removed from the queue manager, and publication flow is not interrupted.

If SubIdentity is specified, but the SubIdentity is not in the identity set for the subscription, then the **Deregister Subscriber** command fails with the return code *MQRCCF_SUB_IDENTITY_ERROR*.

<Filter> (*MQPSC_FILTER*)
The value is a string specifying the filter to be deregistered. It must match exactly, including case and any spaces, a subscription filter that has been previously registered.

This property can, optionally, be repeated if more than one filter is to be deregistered. It can be omitted if a <Topic> is specified, or if DeregAll is specified in <RegOpt>.

The filters specified can be a subset of those registered if the subscriber wants to retain subscriptions for other filters.

<RegOpt> (*MQPSC_REGISTRATION_OPTION*)
The registration options property can take the following values:

DeregAll
(*MQPSC_DEREGISTER_ALL*)

All matching subscriptions registered for this subscriber are to be deregistered.

If you specify DeregAll:
- <Topic>, <SubPoint>, and <Filter> can be omitted.
- <Topic> and <Filter> can be repeated, if required.
- <SubPoint> must not be repeated.

If you do **not** specify DeregAll:
- <Topic> must be specified, and can be repeated if required.
- <SubPoint> and <Filter> can be omitted.
- <SubPoint> must not be repeated.
- <Filter> can be repeated, if required.

If topics and filters are both repeated, then all subscriptions matching all combinations of the two are removed. For example, a **Deregister Subscriber** command that specifies three topics and three filters will attempt to remove nine subscriptions.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The CorrelId in the message descriptor (MQMD), which must not be zero, is used to identify the subscriber. It must match the CorrelId used in the original subscription.

FullResp
(*MQPSC_FULL_RESPONSE*)

WhenFullResp is specified all attributes of the subscription are returned in the response message, if the command does not fail.

When FullResp is specified DeregAll is not permitted in the **Deregister Subscriber** command. It is also not possible to specify multiple topics. The command fails with return code *MQRCCF_REG_OPTIONS_ERROR*, in both cases.

LeaveOnly
(*MQPSC_LEAVE_ONLY*)

When you specify this with a SubIdentity which is in the identity set for the subscription the SubIdentity is removed from the identity set for the subscription. The subscription is not removed from the queue manager, even if the resulting identity set is empty. If the SubIdentity value is not in the identity set the command fails with return code *MQRCCF_SUB_IDENTITY_ERROR*.

If LeaveOnly is specified with no SubIdentity, the command fails with return code *MQRCCF_REG_OPTIONS_ERROR*.

If neither LeaveOnly nor a SubIdentity are specified, then the subscription is removed regardless of the contents of the identity set for the subscription.

None (MQPSC_NONE)

All options take their default values. This has the same effect as omitting the registration options property. If other options are specified at the same time, None is ignored.

VariableUserId
(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager and correlid) is not restricted to a single userid. This differs from the existing behavior of the queue manager that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity, the return code *MQRCCF_DUPLICATE_SUBSCRIPTION* is returned.

Any user can modify or deregister the subscription when they have suitable authority, avoiding the existing check that the userid must match that of the original subscriber.

To add this option to an existing subscription the command must come from the same userid as the original subscription itself.

If the subscription to be deregistered has VariableUserId set this must be set at deregister time to indicate which subscription is being deregistered. Otherwise, the userid of the **Deregister Subscriber** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

The default, if this property is omitted, is that no registration options are set.

<QMgrName> (*MQPSC_Q_MGR_NAME*)
The value is the queue manager name for the subscriber queue. It must match the QMgrName used in the original subscription.

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the queue manager.

<QName> (*MQPSC_Q_NAME*)
The value is the name of the subscriber queue. It must match the QName used in the original subscription.

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank.

<SubName> (*MQPSC_SUBSCRIPTION_NAME*)
If you specify SubName on a **Deregister Subscriber** command the SubName value takes precedence

over all other identifier fields except the userid, unless VariableUserId is set on the subscription itself. If VariableUserId is not set, the **Deregister Subscriber** command succeeds only if the userid of the command message matches that of the subscription, if not the command fails with return code *MQRCCF_DUPLICATE_IDENTITY*.

If a subscription exists that matches the traditional identity of this command but has no SubName the **Deregister Subscriber** command fails with return code *MQRCCF_SUB_NAME_ERROR*. If an attempt is made to deregister a subscription that has a SubName using a command message that matches the traditional identity but with no SubName specified the command succeeds.

<SubUserData> (*MQPSC_SUBSCRIPTION_USER_DATA*)
This is a variable-length text string. The value is stored by the queue manager with the subscription but has no influence on the delivery of the publication to the subscriber. The value can be altered by re-registering to the same subscription with a new value. This attribute is for the use of the application.

SubUserData is returned in the Metatopic information (MQCACF_REG_SUB_USER_DATA) for a subscription, if SubUserData is present.

Example:
Here is an example of NameValueData for a **Deregister Subscriber** command message. In this example, the sample application is deregistering its subscription to the topics which contain the latest score for all matches. The subscriber's identity, including the CorrelId, is taken from the defaults in the MQMD.

```
<psc>
 <Command>DeregSub</Command>
 <RegOpt>CorrelAsId</RegOpt>
 <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Publish message:

The **Publish** command message is put to a queue, or from a queue manager to a subscriber, to publish information on a specified topic or topics.

Authority to put a message onto a queue and authority to publish information on a specified topic or topics is necessary.

If the user has authority to publish information on some, but not all, topics, only those topics are used to publish; a warning response indicates which topics are not used to publish.

If a subscriber has any matching subscriptions, the queue manager forwards the **Publish** message to the subscriber queues defined in the corresponding **Register Subscriber** command messages.

See Queue Manager Response message for details of the message descriptor (MQMD) parameters needed when sending a command message to the queue manager, and used when a queue manager forwards a publication to a subscriber.

The queue manager forwards the **Publish** message to other queue managers in the network that have matching subscriptions, unless it is a local publication.

Publication data, if any, is included in the body of the message. The data can be described in an <mcd> folder in the NameValueData field of the MQRFH2 header.

Properties

<Command> (*MQPSC_COMMAND*)
The value is Publish(*MQPSC_PUBLISH*).

This property must be specified.

<Topic> (*MQPSC_TOPIC*)
The value is a string that contains a topic that categorizes this publication. No wildcard characters are allowed.

You must add the topic to the namelist SYSTEM.QPUBSUB.QUEUE.NAMELIST, see Adding a stream for instructions on how to complete this task.

This property must be specified, and can optionally be repeated for as many topics as needed.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT*)
The subscription point on which the publication is published.

In WebSphere Event Broker V6, the value of the `<SubPoint>` property is the value of the Subscription Point attribute of the Publication node that is handling the publishing.

In WebSphere MQ V7.0.1, the value of the `<SubPoint>` property must match the name of a subscription point. See Adding a subscription point.

<PubOpt> (*MQPSC_PUBLICATION_OPTION*)
The publication options property can take the following values:

RetainPub
(*MQPSC_RETAIN_PUB*)

The queue manager is to retain a copy of the publication. If this option is not set, the publication is deleted as soon as the queue manager has sent the publication to all its current subscribers.

IsRetainedPub
(*MQPSC_IS_RETAINED_PUB*)

(Can only be set by a queue manager.) This publication has been retained by the queue manager. The queue manager sets this option to notify a subscriber that this publication was published earlier and has been retained, provided that the subscription has been registered with the `InformIfRetained` option. It is set only in response to a **Register Subscriber** or **Request Update** command message. Retained publications that are sent directly to subscribers do not have this option set.

Local
(*MQPSC_LOCAL*)

This option tells the queue manager that this publication must not be sent to other queue managers. All subscribers that registered at this queue manager receive this publication if they have matching subscriptions.

OtherSubsOnly
(*MQPSC_OTHER_SUBS_ONLY*)

This option allows simpler processing of conference-type applications, where a publisher is also a subscriber to the same topic. It tells the queue manager not to send the publication to the publisher's subscriber queue even if it has a matching subscription. The publisher's subscriber queue consists of its `QMgrName`, `QName`, and optional `CorrelId`, as described in the following list.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The `CorrelId` in the MQMD (which must not be zero) is part of the publisher's subscriber queue, in applications where the publisher is also a subscriber.

None (*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the publication options property. If other options are specified at the same time, `None` is ignored.

You can have more than one publication option by introducing additional <PubOpt> elements.

The default, if this property is omitted, is that no publication options are set.

<PubTime> (*MQPSC_PUBLISH_TIMESTAMP*)
The value is an optional publication timestamp set by the publisher. It is 16 characters long with format:

```
YYYYMMDDHHMMSSTH
```

using Universal Time. This information is not checked by the queue manager before being sent to the subscribers.

<SeqNum> (*MQPSC_SEQUENCE_NUMBER*)
The value is an optional sequence number set by the publisher.

It must be incremented by 1 with each publication. However, this is not checked by the queue manager, which merely transmits this information to subscribers.

If publications on the same topic are published to different interconnected queue managers, it is the responsibility of the publishers to ensure that sequence numbers, if used, are meaningful.

<QMgrName> (*MQPSC_Q_MGR_NAME*)
The value is a string containing the name of the queue manager for the publisher's subscriber queue, in applications where the publisher is also a subscriber (see OtherSubsOnly).

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the name of the queue manager.

<QName> (*MQPSC_Q_NAME*)
The value is a string containing the name of the publisher's subscriber queue, in applications where the publisher is also a subscriber (see OtherSubsOnly).

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank if OtherSubsOnly is set.

Example

Here are some examples of *NameValueData* for a **Publish** command message.

The first example is for a publication sent by the match simulator in the sample application to indicate that a match has started.

```
<psc>
 <Command>Publish</Command>
 <Topic>Sport/Soccer/Event/MatchStarted</Topic>
</psc>
```

The second example is for a retained publication. The latest score in the match between Team1 and Team2 is published.

```
<psc>
 <Command>Publish</Command>
 <PubOpt>RetainPub</PubOpt>
 <Topic>Sport/Soccer/State/LatestScore/Team1 Team2</Topic>
</psc>
```

Register Subscriber message:

The **Register Subscriber** command message is sent to a queue manager by a subscriber, or by another application on behalf of a subscriber, to indicate that it wants to subscribe to one or more topics at a subscription point. A message content filter can also be specified.

In publish/subscribe filter expressions, nesting parentheses causes performance to decrease exponentially. Avoid nesting parentheses to a depth greater than about 6.

The message is sent to SYSTEM.BROKER.CONTROL.QUEUE, which is the queue manager's control queue. Authority to put a message to this queue is required, in addition to access authority (set by the queue manager's system administrator) for the topic, or topics, in the subscription.

If the user has authority on some, but not all, topics, only those with authority are registered; a warning response indicates those that are not registered.

See "MQMD settings in command messages to the queue manager" on page 2201 for details of the message descriptor (MQMD) parameters that are needed when sending a command message to the queue manager.

If the reply to queue is a temporary dynamic queue, the subscription is deregistered automatically by the queue manager when the queue is closed.

Properties

<Command> (*MQPSC_COMMAND*)
The value is RegSub (*MQPSC_REGISTER_SUBSCRIBER*). This property must be specified.

<Topic> (*MQPSC_TOPIC*)
The topic for which the subscriber wants to receive publications. Wildcard characters can be specified as part of the topic.

If you use the MQSC command **display sub** to examine the subscription created in this way, the value of the <Topic> tag is shown as the TOPICSTR property of the subscription.

This property is required, and can optionally be repeated for as many topics as needed.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT*)
The value is the subscription point to which the subscription is attached.

If this property is omitted, the default subscription point is used.

In WebSphere Event Broker V6, the value of the <SubPoint> property must match the value of the Subscription Point attribute of the Publication nodes that are subscribed to.

In WebSphere MQ V7.0.1, the value of the <SubPoint> property must match the name of a subscription point. See Adding a subscription point .

<Filter> (*MQPSC_FILTER*)
The value is an SQL expression that is used as a filter on the contents of publication messages. If a publication on the specified topic matches the filter, it is sent to the subscriber. This property corresponds to the Selection String that is used in MQSUB and MQOPEN calls. For more information, see Selecting on the content of a message

If this property is omitted, no content filtering takes place.

<RegOpt> (*MQPSC_REGISTRATION_OPTION*)
This Registration Options property can take the following values:

AddName
(*MQPSC_ADD_NAME*)

When specified for an existing subscription that matches the traditional identity of this Register Subscription command, but with no current SubName value, the SubName specified in this command is added to the subscription.

If AddName is specified the SubName field is mandatory, otherwise MQRCCF_REG_OPTIONS_ERROR is returned.

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The CorrelId in the message descriptor (MQMD) is used when sending matching publications to the subscriber queue. The CorrelId must not be zero,

FullResp
(*MQPSC_FULL_RESPONSE*)

When specified all attributes of the subscription are returned in the response message, if the command does not fail.

FullResp is valid only when the command message refers to a single subscription. Therefore, only one topic is permitted in the command; otherwise the command fails with return code *MQRCCF_REG_OPTIONS_ERROR*.

InformIfRet
(*MQPSC_INFORM_IF_RETAINED*)

The queue manager informs the subscriber if a publication is retained when it sends a Publish message in response to a **Register Subscriber** or **Request Update** command message. The queue manager does this by including the IsRetainedPub publication option in the message.

JoinExcl
(*MQPSC_JOIN_EXCLUSIVE*)

This option indicates that the specified SubIdentity should be added as the exclusive member of the identity set for the subscription, and that no other identities can be added to the set.

If the identity has already joined 'shared' and is the sole entry in the set, the set is changed to an exclusive lock held by this identity. Otherwise, if the subscription currently has other identities in the identity set (with shared access) the command fails with return code *MQRCCF_SUBSCRIPTION_IN_USE*.

JoinShared
(*MQPSC_JOIN_SHARED*)

This option indicates that the specified SubIdentity should be added to the identity set for the subscription.

If the subscription is currently locked exclusively (using the JoinExcl option), the command fails with return code *MQRCCF_SUBSCRIPTION_LOCKED*, unless the identity that has the subscription locked is the same identity as that in this command message. In this case the lock is automatically modified to a shared lock.

Local
(*MQPSC_LOCAL*)

The subscription is local and is not distributed to other queue managers in the network. Publications made at other queue managers are not delivered to this subscriber, unless it also has a corresponding global subscription.

NewPubsOnly
(*MQPSC_NEW_PUBS_ONLY*)

Retained publications that exist at the time the subscription is registered are not sent to the subscriber; only new publications are sent.

If a subscriber re-registers and changes this option so that it is no longer set, a publication that has already been sent to it might be sent again.

NoAlter
(*MQPSC_NO_ALTER*)

The attributes of an existing matching subscription is not changed.

When a subscription is being created, this option is ignored. All other options specified apply to the new subscription.

If a `SubIdentity` also has one of the join options (`JoinExcl` or `JoinShared`) specified, the identity is added to the identity set regardless of whether `NoAlter` is specified.

None (*MQPSC_NONE*)

All registration options take their default values.

If the subscriber is already registered, its options are reset to their default values (note that this does *not* have the same affect as omitting the registration options property), and the subscription expiry is updated from the MQMD of the **Register Subscriber** message.

If other registration options are specified at the same time, `None` is ignored.

NonPers
(*MQPSC_NON_PERSISTENT*)

Publications matching this subscription are delivered to the subscriber as non-persistent messages.

Pers (*MQPSC_PERSISTENT*)

Publications matching this subscription are delivered to the subscriber as persistent messages.

PersAsPub
(*MQPSC_PERSISTENT_AS_PUBLISH*)

Publications matching this subscription are delivered to the subscriber with the persistence specified by the publisher. This is the default behavior.

PersAsQueue
(*MQPSC_PERSISTENT_AS_Q*)

Publications matching this subscription are delivered to the subscriber with the persistence specified on the subscriber queue.

PubOnReqOnly
(*MQPSC_PUB_ON_REQUEST_ONLY*)

The queue manager does not send publications to the subscriber, except in response to a **Request Update** command message.

VariableUserId
(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager and correlid) is not restricted to a single userid. This differs from the existing behavior of the queue manager that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity *MQRCCF_DUPLICATE_SUBSCRIPTION* is returned.

This allows any user to modify or deregister the subscription if the user has suitable authority. There is therefore no need to check that the userid matches that of the original subscriber.

To add this option to an existing subscription the command must come from the same userid as the original subscription itself.

If the subscription of the **Request Update** command has VariableUserId set, this must be set at request update time to indicate which subscription is referred to. Otherwise, the userid of the **Request Update** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

If a **Register Subscriber** command message without this option set refers to an existing subscription which has this option set, the option is removed from this subscription and the userid of the subscription is now fixed. If there already exists a subscriber which has the same identity (queue, queue manager and correlation identifier) but with a different user ID associated to it, the command fails with return code *MQRCCF_DUPLICATE_IDENTITY* because there can only be one userid associated with a subscriber identity.

If the registration options property is omitted and the subscriber is already registered, its registration options are not changed and the subscription expiry is updated from the MQMD of the **Register Subscriber** message.

If the subscriber is not already registered, a new subscription is created with all registration options taking their default values.

The default values are PersAsPub and no other options set.

<QMgrName> (*MQPSC_Q_MGR_NAME*)
The value is the name of the queue manager for the subscriber queue, to which matching publications are sent by the queue manager.

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the queue manager's QMgrName.

<QName> (*MQPSC_Q_NAME*)
The value is the name of the subscriber queue, to which matching publications are sent by the queue manager.

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank in this case.

If the queue is a temporary dynamic queue, nonpersistent delivery of publications (NonPers) must be specified in the <RegOpt> property.

If the queue is a temporary dynamic queue, the subscription is deregistered automatically by the queue manager when the queue is closed.

<SubName> (*MQPSC_SUBSCRIPTION_NAME*)
This is a name given to a particular subscription. You can use it instead of the queue manager, queue and optional correlId to refer to a subscription.

If a subscription already exists with this **SubName**, any other attributes of the subscription (Topic, QMgrName, QName, CorrelId, UserId, RegOpts, UserSubData, and Expiry) are overridden with the attributes, if specified, that are passed in the new **Register Subscriber** command message. However, if **SubName** is used with no QName field specified, and a ReplyToQ is specified in the MQMD header, the subscriber queue is changed to be the ReplyToQ.

If a subscription that matches the traditional identity of this command already exists, but has no **SubName**, the Registration command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*, unless the **AddName** option is specified.

If you try to alter an existing named subscription by using another **Register Subscriber** command that specifies the same **SubName**, and the values of Topic, QMgrName, QName, and CorrelId in the new command match a different existing subscription, with or without a SubName defined, the command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*. This prevents two subscription names referring to the same subscription.

<SubIdentity> (*MQPSC_SUBSCRIPTION_IDENTITY*)
This string is used to represent an application with an interest in a subscription. It is a variable-length character string with a maximum length of 64 characters, and is optional. The queue manager maintains a set of subscriber identities for each subscription. Each subscription can allow its identity set to contain only one identity, or an unlimited number of identities (see the **JoinShared** and **JoinExcl** options).

A subscribe command that specifies the **JoinShared** or **JoinExcl** option adds the **SubIdentity** to the subscription's identity set, if it is not already there and if the existing set of identities allows such an action; that is, no other subscriber has joined exclusively or the identity set is empty.

Any alteration of the subscription's attributes as the result of a **Register Subscriber** command in which a **SubIdentity** is specified, only succeeds if it would be the only member of the set of identities for this subscription. Otherwise the command fails with return code *MQRCCF_SUBSCRIPTION_IN_USE*. This prevents a subscription's attributes from changing without other interested subscribers being aware.

If you specify a character string that is longer than 64 characters, the command fails with return code *MQRCCF_SUB_IDENTITY_ERROR*.

<SubUserData> (*MQPSC_SUBSCRIPTION_USER_DATA*)
This is a variable-length text string. The value is stored by the queue manager with the subscription, but has no influence on publication delivery to the subscriber. The value can be altered by re-registering to the same subscription with a new value. This attribute is there for the use of the application.

The **SubUserData** is returned in the Metatopic information (*MQCACF_REG_SUB_USER_DATA*) for a subscription if present.

If you specify more than one of the registration option values `NonPers`, `PersAsPub`, `PersAsQueue`, and `Pers`, then only the last one is used. You cannot combine these options in an individual subscription.

Example

Here is an example of `NameValueData` for a **Register Subscriber** command message. In the sample application, the results service uses this message to register a subscription to the topics containing the latest scores in all matches, with the 'Persistent as publish' option set. The subscriber's identity, including the `CorrelId`, is taken from the defaults in the MQMD.

```
<psc>
 <Command>RegSub</Command>
 <RegOpt>PersAsPub</RegOpt>
 <RegOpt>CorrelAsId</RegOpt>
 <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Related reference:

Publish/Subscribe: Wildcards
Publish/subscribe wildcard schemes have changed. Version 6.0 uses character-based wildcards. Now, you have a choice of using character-based and topic-based wildcards. The IBM WebSphere MQ topic-based wildcard scheme differs slightly from the topic-based wildcard scheme in WebSphere Event Broker version 6.0 and WebSphere Message Broker version 6.0 and 6.1. The IBM WebSphere MQ topic-based scheme allows multilevel wildcards in the middle of a topic string. The WebSphere Message Broker and WebSphere Event Broker scheme permits only topic multilevel wildcards at the end of a topic string.

Request Update message:

The **Request Update** command message is sent from a subscriber to a queue manager, to request the current retained publications for the specified topic and subscription point that match the given (optional) filter.

This message is sent to *SYSTEM.BROKER.CONTROL.QUEUE*, the queue manager's control queue. Authority to put a message to this queue is required, in addition to access authority for the topic in the request update; this is set by the queue manager's system administrator.

This command is normally used if the subscriber specified the option `PubOnReqOnly` when it registered. If the queue manager has any matching retained publications, they are sent to the subscriber. If the queue manager has no matching retained publications, the request fails with return code *MQRCCF_NO_RETAINED_MSG*. The requester must have previously registered a subscription with the same Topic, SubPoint, and Filter values.

Properties:

<Command> (*MQPSC_COMMAND*)
The value is `ReqUpdate` (*MQPSC_REQUEST_UPDATE*). This property must be specified.

<Topic> (*MQPSC_TOPIC*)
The value is the topic that the subscriber is requesting; wildcard characters are allowed.

This property must be specified, but only one occurrence is allowed in this message.

<SubPoint> (*MQPSC_SUBSCRIPTION_POINT*)
The value is the subscription point to which the subscription is attached.

If this property is omitted, the default subscription point is used.

<Filter> (*MQPSC_FILTER*)
The value is an ESQL expression that is used as a filter on the contents of publication messages. If a publication on the specified topic matches the filter, it is sent to the subscriber.

The `<Filter>` property should have the same value as that specified on the original subscription for which you are now requesting an update.

If this property is omitted, no content filtering takes place.

<RegOpt> (*MQPSC_REGISTRATION_OPTION*)
The registration options property can take the following value:

CorrelAsId
(*MQPSC_CORREL_ID_AS_IDENTITY*)

The `CorrelId` in the message descriptor (MQMD), which must not be zero, is used when sending matching publications to the subscriber queue.

None (*MQPSC_NONE*)

All options take their default values. This has the same effect as omitting the `<RegOpt>` property. If other options are specified at the same time, `None` is ignored.

VariableUserId
(*MQPSC_VARIABLE_USER_ID*)

When specified the identity of the subscriber (queue, queue manager, and correlid) is not restricted to a single userid. This differs from the existing behavior of the queue manager that associates the userid of the original registration message with the subscriber's identity and from then on prevents any other user using that identity. If a new subscriber tries to use the same identity, the command fails with return code *MQRCCF_DUPLICATE_SUBSCRIPTION*.

This allows any user to modify or deregister the subscription when they have suitable authority. Therefore, there is no need to check that the userid matches that of the original subscriber.

To add this option to an existing subscription, the command must come from the same userid as the original subscription.

If the subscription of the **Request Update** command has VariableUserId set, this must be set at request update time to indicate which subscription is referred to. Otherwise, the userid of the **Request Update** command is used to identify the subscription. This is overridden, along with the other subscriber identifiers, if a subscription name is supplied.

The default, if this property is omitted, is that no registration options are set.

<QMgrName> (*MQPSC_Q_MGR_NAME*)
The value is the name of the queue manager for the subscriber queue, to which the matching retained publication is sent by the queue manager.

If this property is omitted, the default is the ReplyToQMgr name in the message descriptor (MQMD). If the resulting name is blank, it defaults to the queue manager's QMgrName.

<QName> (*MQPSC_Q_NAME*)
The value is the name of the subscriber queue, to which the matching retained publication is sent by the queue manager.

If this property is omitted, the default is the ReplyToQ name in the message descriptor (MQMD), which must not be blank in this case.

<SubName> (*MQPSC_SUBSCRIPTION_NAME*)
This is a name given to a particular subscription. If specified on a **Request Update** command the SubName value takes precedence over all other identifier fields except the userid, unless VariableUserId is set on the subscription itself. If VariableUserId is not set, the *Request Update* command succeeds only if the userid of the command message matches that of the subscription. If the userid of the command message does not match that of the subscription, the command fails with return code *MQRCCF_DUPLICATE_IDENTITY*.

If VariableUserId is set, and the userid differs from that of the subscription, the command succeeds if the userid of the new command message has authority to browse the stream queue and put to the subscriber queue of the subscription. Otherwise, the command fails with return code *MQRCCF_NOT_AUTHORIZED*.

If a subscription exists that matches the traditional identity of this command, but has no SubName, the**Request Update** command fails with return code *MQRCCF_SUB_NAME_ERROR*.

If an attempt is made to request an update for a subscription that has a SubName using a command message that matches the traditional identity, but with no SubName specified, the command succeeds.

Example:
Here is an example of NameValueData for a **Request Update** command message. In the sample application, the results service uses this message to request retained publications containing the latest scores for all teams. The subscriber's identity, including the CorrelId, is taken from the defaults in the MQMD.

```
<psc>
 <Command>ReqUpdate</Command>
 <RegOpt>CorrelAsId</RegOpt>
 <Topic>Sport/Soccer/State/LatestScore/#</Topic>
</psc>
```

Queue Manager Response message:

A **Queue Manager Response** message is sent from a queue manager to the ReplyToQ of a publisher or a subscriber, to indicate the success or failure of a command message received by the queue manager if the command message descriptor specified that a response is required.

The response message is contained within the NameValueData field of the MQRFH2 header, in a <pscr> folder.

In the case of a warning or error, the response message contains the <psc> folder from the command message as well as the <pscr> folder. The message data, if any, is not contained in the queue manager

response message. In the case of an error, none of the message that caused an error has been processed; in the case of a warning, some of the message might have been processed successfully.

If there is a failure sending a response:

- For publication messages, the queue manager tries to send the response to the WebSphere MQ dead-letter queue if the MQPUT fails. This allows the publication to be sent to subscribers even if the response cannot be sent back to the publisher.
- For other messages, or if the publication response cannot be sent to the dead-letter queue, an error is logged and the command message is normally rolled back. Whether this happens depends on how the MQInput node has been configured.

Properties:

<Completion> (*MQPSCR_COMPLETION*)
The completion code, which can take one of three values:
ok Command completed successfully
warning
Command completed but with warning
error Command failed

<Response> (*MQPSCR_RESPONSE*)
The response to a command message, if that command produced a completion code of `warning` or `error`. It contains a `<Reason>` property, and might contain other properties that indicate the cause of the warning or error.

In the case of one or more errors, there is only one response folder, indicating the cause of the first error only. In the case of one or more warnings, there is a response folder for each warning.

<Reason> (*MQPSCR_REASON*)
The reason code qualifying the completion code, if the completion code is a `warning` or `error`. It is set to one of the error codes listed in the following example. The `<Reason>` property is contained within a `<Response>` folder. The reason code can be followed by any valid property from the `<psc>` folder (for example, a topic name), indicating the cause of the error or warning. If you get a reason code of ????, check the data for correctness, for example, matching angled brackets (< >).

Examples:
Here are some examples of `NameValueData` in a **Queue Manager Response** message. A successful response might be the following:

```
<pscr>
  <Completion>ok</Completion>
</pscr>
```

Here is an example of a failure response; the failure is a filter error. The first `NameValueData` string contains the response; the second contains the original command.

```
<pscr>
  <Completion>error</Completion>
  <Response>
    <Reason>3150</Reason>
  </Reponse>
</pscr>

<psc>
 ...
 command message (to which
 the queue manager is responding)
 ...
</psc>
```

Here is an example of a warning response (due to unauthorized topics). The first `NameValueData` string contains the response; the second `NameValueData` string contains the original command.

```
<pscr>
  <Completion>warning</Completion>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic1</Topic>
  </Reponse>
  <Response>
    <Reason>3081</Reason>
    <Topic>topic2</Topic>
  </Reponse>
</pscr>

<psc>
 ...
 command message (to which
 the queue manager is responding)
 ...
</psc>
```

Publish/subscribe reason codes:

These reason codes might be returned in the `Reason` field of a publish/subscribe response `<pscr>` folder. Constants that can be used to represent these codes in the C or C++ programming languages are also listed.

The MQRC_ constants require the WebSphere MQ `cmqc.h` header file. The MQRCCF_ constants require the WebSphere MQ `cmqcfc.h` header file (apart from *MQRCCF_FILTER_ERROR* and *MQRCCF_WRONG_USER*, which require the `cmqpsc.h` header file).

Reason code and text	Explanation	Issued by
2336 MQRC_RFH_COMMAND_ERROR	Valid values for the `<Command>` field of a `<psc>` folder are: RegSub, DeregSub, Publish, DeletePub, and ReqUpdate. Any other values result in this error code being issued.	Any command
2337 MQRC_RFH_ PARM_ERROR	The `<psc>` and `<mcd>` folders both have a set of valid parameters that can be specified within them. Check the descriptions of these folders and ensure that you have not specified incorrect parameters.	Any command
2338 MQRC_RFH_DUPLICATE_PARM	Some parameters (for example, Topic) within a `<psc>` folder can be repeated, but others (for example, Command) cannot be repeated. Check that you have not duplicated a non-repeatable parameter.	Any command
2339 MQRC_RFH_PARM_MISSING	Some parameters within `<psc>` or `<mcd>` folders are optional and can be omitted; some are mandatory and must not be omitted. Check that you have included all mandatory parameters within your `<psc>` and `<mcd>` folders.	Any command

Reason code and text	Explanation	Issued by
2551 MQRC_SELECTION_NOT_AVAILABLE	No extended message selection provider was available to determine which subscribers with a filter specified should receive the publication.	Publish, Register Subscriber, and Request Update
	No extended message selection provider was available to handle the filter of the specified subscriber.	Register Subscriber and Request Update
2554 MQRC_CONTENT_ERROR	An extended message selection provider found an error in the current or retained publication.	Publish and Request Update
3008 MQRCCF_COMMAND_FAILED	An internal error occurred which prevented the command from executing correctly. The error might occur if the command is reissued. The system event log for the queue manager contains information which should be used when reporting the problem to IBM.	Any command
3072 MQRCCF_TOPIC_ERROR	One or more of the values you supplied for the Topic parameter are incorrect. Check that your values for Topic conform to the specified restrictions.	Any command
3073 MQRCCF_NOT_REGISTERED	The combination of SubPoint, Topic, and Filter that you specified on your DeregSub or ReqUpdate command was either not a combination with which you had previously registered or, for the DeregSub command if the DeregAll option was specified, one of the SubPoint, Topic, or Filter properties was not used to deregister any subscription.	Deregister Subscriber and Request Update commands
3074 MQRCCF_Q_MGR_NAME_ERROR	The specified queue manager was not valid, or the queue manager was not available or did not exist.	Deregister Subscriber, Publish, Register Subscriber, and Request Update commands
3076 MQRCCF_Q_NAME_ERROR	The specified queue name was not valid, or the queue did not exist on the specified queue manager.	Deregister Subscriber, Publish, Register Subscriber, and Request Update commands
3077 MQRCCF_NO_RETAINED_MSG	There were no retained messages for the topic you specified. This might or might not be an error, depending on the design of your application program.	Request Update command
3079 MQRCCF_INCORRECT_Q	RegSub, DeregSub, and ReqUpdate commands are always sent to the SYSTEM.BROKER.CONTROL.QUEUE queue of the queue manager for which they are intended. Publish and Delete Publication commands are sent to the input queue for the particular publish/subscribe message flow for which they are intended; this is determined when the message flow is designed. This error code is returned if a command is sent to the wrong queue.	Any command

Reason code and text	Explanation	Issued by
3080 MQRCCF_CORREL_ID_ERROR	You have specified CorrelAsId as one of your RegOpt parameters. However, the CorrelId field of the MQMD does not contain a valid correlation identifier (that is, it is set to MQCI_NONE).	Deregister Subscriber and Register Subscriber commands
3081 MQRCCF_NOT_AUTHORIZED	You are not authorized to perform the requested action. Authorization settings for the queue manager are handled by the system administrator using the Topics Hierarchy editor.	Publish and Register Subscriber commands
3083 MQRCCF_REG_OPTIONS_ERROR	You have specified an unrecognized RegOpt parameter in the <psc> folder that contains your RegSub or DeregSub command.	Deregister Subscriber and Register Subscriber commands
3084 MQRCCF_PUB_OPTIONS_ERROR	You have specified an unrecognized PubOpt parameter in the <psc> folder that contains your Publish command.	Publish command
3087 MQRCCF_DEL_OPTIONS_ERROR	You have specified an unrecognized DelOpt parameter in the <psc> folder that contains your DeletePub command.	Delete Publication command
3150 MQRCCF_FILTER_ERROR	The value specified for the Filter parameter is not valid. Check the section that describes the valid syntax for filter expressions and ensure that your expression conforms.	Deregister Subscriber, Register Subscriber, and Request Update commands
3151 MQRCCF_WRONG_USER	A subscription that matches the one specified already exists; however, it was registered by a different user. A subscription can only be changed or deregistered by the user who originally registered it.	Deregister Subscriber, Register Subscriber, and Request Update commands
3152 MQRCCF_DUPLICATE_SUBSCRIPTION	A matching subscription already exists with a different subscription name.	
3153 MQRCCF_SUB_NAME_ERROR	Either the format of the subscription name is not valid, or a matching subscription already exists with no subscription name.	
3154 MQRCCF_SUB_IDENTITY_ERROR	The subscription identity parameter is in error. Either the supplied value exceeds the maximum length allowed, or the subscription identity is not currently a member of the subscription's identity set and a Join registration option was not specified.	
3155 MQRCCF_SUBSCRIPTION_IN_USE	An attempt to modify or deregister a subscription was attempted by a member of the identity set when it was not the only member of this set.	
3156 MQRCCF_SUBSCRIPTION_LOCKED	The subscription is currently exclusively locked by another identity.	

Reason code and text	Explanation	Issued by
3157 MQRCCF_ALREADY_JOINED	A Join registration option was specified but the subscriber identity was already a member of the subscription's identity set.	

MQMD settings in command messages to the queue manager:

Applications that send command messages to the queue manager use the following settings of fields in the message descriptor (MQMD). Fields that are left as the default value, or can be set to any valid value in the usual way, are not listed here.

Report
See MsgType and CorrelId.

MsgType
MsgType should be set to either *MQMT_REQUEST* or *MQMT_DATAGRAM*. *MQRC_MSG_TYPE_ERROR* will be returned if MsgType is not set to one of these values.

MsgType should be set to *MQMT_REQUEST* for a command message if a response is always required. The MQRO_PAN and MQRO_NAN flags in the Report field are not significant in this case.

If MsgType is set to *MQMT_DATAGRAM*, responses depend on the setting of the MQRO_PAN and MQRO_NAN flags in the Report field:

- MQRO_PAN alone means that the queue manager sends a response only if the command succeeds.
- MQRO_NAN alone means that the queue manager sends a response only if the command fails.
- If a command completes with a warning, a response is sent if either MQRO_PAN or MQRO_NAN is set.
- MQRO_PAN + MQRO_NAN means that the queue manager sends a response whether the command succeeds or fails. This has the same effect from the queue manager's perspective as setting MsgType to MQMT_REQUEST.
- If neither MQRO_PAN nor MQRO_NAN is set, no response is ever sent.

Format
Set to MQFMT_RF_HEADER_2

MsgId
This field is normally set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId
This field can be set to any value. If the sender's identity includes a CorrelId, specify this value, together with MQRO_PASS_CORREL_ID in the Report field, to ensure that it is set in all response messages sent by the queue manager to the sender.

ReplyToQ
This field defines the queue to which responses, if any, are to be sent. This might be the sender's queue; this has the advantage that the QName parameter can be omitted from the message. If, however, responses are to be sent to a different queue, the QName parameter is needed.

ReplyToQMgr
This field defines the queue manager for responses. If you leave this field blank (the default value), the local queue manager puts its own name in this field.

MQMD settings for publications forwarded by a queue manager:

A queue manager uses these settings of fields in the message descriptor (MQMD) when it sends a publication to a subscriber. All other fields in the MQMD are set to their default values.

Report
Report is set to MQRO_NONE.

MsgType
MsgType is set to MQMT_DATAGRAM.

Expiry
Expiry is set to the value in the **Publish** message received from the publisher. In the case of a retained message, the time outstanding is reduced by the approximate time that the message has been at the queue manager.

Format
Format is set to MQFMT_RF_HEADER_2

MsgId
MsgId is set to a unique value.

CorrelId
If CorrelId is part of the subscriber's identity, this is the value specified by the subscriber when registering. Otherwise, it is a non-zero value chosen by the queue manager.

Priority
Priority takes the value set by the publisher, or as resolved if the publisher specified MQPRI_PRIORITY_AS_Q_DEF.

Persistence
Persistence takes the value set by the publisher, or as resolved if the publisher specified MQPER_PERSISTENCE_AS_Q_DEF, unless specified otherwise in the **Register Subscriber** message for the subscriber to which this publication is being sent.

ReplyToQ
ReplyToQ is set to blanks.

ReplyToQMgr
ReplyToQMgr is set to the name of the queue manager.

UserIdentifier
UserIdentifier is the subscriber's user identifier, as set when the subscriber registered.

AccountingToken
AccountingToken is the subscriber's accounting token, as set when the subscriber first registered.

ApplIdentityData
ApplIdentityData is the subscriber's application identity data, as set when the subscriber first registered.

PutApplType
PutApplType is set to MQAT_BROKER.

PutApplName
PutApplName is set to the first 28 characters of the name of the queue manager.

PutDate
PutDate is the date when the message was put.

PutTime
PutTime is the time when the message was put.

ApplOriginData
ApplOriginData is set to blanks.

MQMD settings in queue manager response messages:

A queue manager uses these settings of fields in the message descriptor (MQMD) when sending a reply to a publication message. All other fields in the MQMD are set to their default values.

Report
Report is set to all zeros.

MsgType
MsgType is set to MQMT_REPLY.

Format
Format is set to MQFMT_RF_HEADER_2

MsgId
The setting of MsgId depends on the Report options in the original command message. By default, it is set to MQMI_NONE, so that the queue manager generates a unique value.

CorrelId
The setting of CorrelId depends on the Report options in the original command message. By default, this means that the CorrelId is set to the same value as the MsgId of the command message. This can be used to correlate commands with their responses.

Priority
Priorityis set to the same value as in the original command message.

Persistence
Persistence is set to the value set in the original command message.

Expiry
Expiry is set to the same value as in the original command message received by the queue manager.

PutApplType
PutApplType is set to MQAT_BROKER.

PutApplName
PutApplName is set to the first 28 characters of name of the queue manager.

Other context fields are set as if generated with MQPMO_PASS_IDENTITY_CONTEXT.

Machine encodings

This section describes the structure of the *Encoding* field in the message descriptor.

See "MQMD - Message descriptor" on page 1705 for a summary of the fields in the structure.

The *Encoding* field is a 32-bit integer that is divided into four separate subfields; these subfields identify:

- The encoding used for binary integers
- The encoding used for packed-decimal integers
- The encoding used for floating-point numbers
- Reserved bits

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined:

MQENC_INTEGER_MASK
Mask for binary-integer encoding.

This subfield occupies bit positions 28 through 31 within the *Encoding* field.

MQENC_DECIMAL_MASK
Mask for packed-decimal-integer encoding.

This subfield occupies bit positions 24 through 27 within the *Encoding* field.

MQENC_FLOAT_MASK
Mask for floating-point encoding.

This subfield occupies bit positions 20 through 23 within the *Encoding* field.

MQENC_RESERVED_MASK
Mask for reserved bits.

This subfield occupies bit positions 0 through 19 within the *Encoding* field.

Binary-integer encoding:

The following values are valid for the binary-integer encoding:

MQENC_INTEGER_UNDEFINED
Binary integers are represented using an encoding that is undefined.

MQENC_INTEGER_NORMAL
Binary integers are represented in the conventional way:

- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address

MQENC_INTEGER_REVERSED
Binary integers are represented in the same way as MQENC_INTEGER_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_INTEGER_NORMAL.

Packed-decimal-integer encoding:

The following values are valid for the packed-decimal-integer encoding:

MQENC_DECIMAL_UNDEFINED
Packed-decimal integers are represented using an encoding that is undefined.

MQENC_DECIMAL_NORMAL
Packed-decimal integers are represented in the conventional way:

- Each decimal digit in the printable form of the number is represented in packed decimal by a single hexadecimal digit in the range X'0' through X'9'. Each hexadecimal digit occupies four bits, and so each byte in the packed decimal number represents two decimal digits in the printable form of the number.
- The least significant byte in the packed-decimal number is the byte that contains the least significant decimal digit. Within that byte, the most significant four bits contain the least significant decimal digit, and the least significant four bits contain the sign. The sign is either X'C' (positive), X'D' (negative), or X'F' (unsigned).
- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address.
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address.

MQENC_DECIMAL_REVERSED
Packed-decimal integers are represented in the same way as MQENC_DECIMAL_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_DECIMAL_NORMAL.

Floating-point encoding:

The following values are valid for the floating-point encoding:

MQENC_FLOAT_UNDEFINED
Floating-point numbers are represented using an encoding that is undefined.

MQENC_FLOAT_IEEE_NORMAL
Floating-point numbers are represented using the standard IEEE[2] floating-point format, with the bytes arranged as follows:

- The least significant byte in the mantissa has the highest address of any of the bytes in the number; the byte containing the exponent has the lowest address
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address

Details of the IEEE float encoding can be found in IEEE Standard 754.

MQENC_FLOAT_IEEE_REVERSED
Floating-point numbers are represented in the same way as MQENC_FLOAT_IEEE_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_FLOAT_IEEE_NORMAL.

MQENC_FLOAT_S390
Floating-point numbers are represented using the standard System/390 floating-point format; this is also used by System/370.

Constructing encodings:

To construct a value for the *Encoding* field in MQMD, the relevant constants that describe the required encodings can be:

- Added together, or
- Combined using the bitwise OR operation (if the programming language supports bit operations)

Whichever method is used, combine only one of the MQENC_INTEGER_* encodings with one of the MQENC_DECIMAL_* encodings and one of the MQENC_FLOAT_* encodings.

Analyzing encodings:

The *Encoding* field contains subfields; because of this, applications that need to examine the integer, packed decimal, or float encoding must use one of the techniques described.

Using bit operations

If the programming language supports bit operations, perform the following steps:

1. Select one of the following values, according to the type of encoding required:
   - MQENC_INTEGER_MASK for the binary integer encoding
   - MQENC_DECIMAL_MASK for the packed decimal integer encoding
   - MQENC_FLOAT_MASK for the floating point encoding

   Call the value `A`.
2. Combine the *Encoding* field with `A` using the bitwise AND operation; call the result `B`.
3. `B` is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

2. The Institute of Electrical and Electronics Engineers

Using arithmetic

If the programming language *does not* support bit operations, perform the following steps using integer arithmetic:

1. Select one of the following values, according to the type of encoding required:
   - 1 for the binary integer encoding
   - 16 for the packed decimal integer encoding
   - 256 for the floating point encoding

   Call the value `A`.
2. Divide the value of the *`Encoding`* field by `A`; call the result `B`.
3. Divide `B` by 16; call the result `C`.
4. Multiply `C` by 16 and subtract from `B`; call the result `D`.
5. Multiply `D` by `A`; call the result `E`.
6. `E` is the encoding required, and can be tested for equality with each of the values that is valid for that type of encoding.

Summary of machine architecture encodings:

Encodings for machine architectures are shown in Table 225.

Table 225. Summary of encodings for machine architectures

Machine architecture	Binary integer encoding	Packed-decimal integer encoding	Floating-point encoding
IBM i	normal	normal	IEEE normal
Intel x86	reversed	reversed	IEEE reversed
PowerPC	normal	normal	IEEE normal
System/390	normal	normal	System/390

Report options and message flags

This section describes the *`Report`* and *`MsgFlags`* fields that are part of the message descriptor MQMD specified on the MQGET, MQPUT, and MQPUT1 calls.

The topics in this section describe:

- The structure of the report field and how the queue manager processes it
- How an application analyzes the report field
- The structure of the message-flags field

For more information about the MQMD message descriptor, see "MQMD - Message descriptor" on page 1705.

Structure of the report field:

This information describes the structure of the report field.

The *`Report`* field is a 32-bit integer that is divided into three separate subfields. These subfields identify:

- Report options that are rejected if the local queue manager does not recognize them
- Report options that are always accepted, even if the local queue manager does not recognize them
- Report options that are accepted only if certain other conditions are satisfied

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits in a subfield are not necessarily adjacent. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MQRO_REJECT_UNSUP_MASK
This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_REPORT_OPTIONS_ERROR.

This subfield occupies bit positions 3, and 11 through 13.

MQRO_ACCEPT_UNSUP_MASK
This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls. Completion code MQCC_WARNING with reason code MQRC_UNKNOWN_REPORT_OPTION are returned in this case.

This subfield occupies bit positions 0 through 2, 4 through 10, and 24 through 31.

The following report options are included in this subfield:

- MQRO_ACTIVITY
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_DEAD_LETTER_Q
- MQRO_DISCARD_MSG
- MQRO_EXCEPTION
- MQRO_EXCEPTION_WITH_DATA
- MQRO_EXCEPTION_WITH_FULL_DATA
- MQRO_EXPIRATION
- MQRO_EXPIRATION_WITH_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA
- MQRO_NAN
- MQRO_NEW_MSG_ID
- MQRO_NONE
- MQRO_PAN
- MQRO_PASS_CORREL_ID
- MQRO_PASS_MSG_ID

MQRO_ACCEPT_UNSUP_IF_XMIT_MASK
This mask identifies the bit positions within the *Report* field where report options that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ObjectQMgrName* and *ObjectName* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code MQCC_WARNING with reason code MQRC_UNKNOWN_REPORT_OPTION are returned if these conditions are satisfied, and MQCC_FAILED with reason code MQRC_REPORT_OPTIONS_ERROR if not.

This subfield occupies bit positions 14 through 23.

The following report options are included in this subfield:

- MQRO_COA

- MQRO_COA_WITH_DATA
- MQRO_COA_WITH_FULL_DATA
- MQRO_COD
- MQRO_COD_WITH_DATA
- MQRO_COD_WITH_FULL_DATA

If any options are specified in the *Report* field that the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *Report* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described above are returned.

If MQCC_WARNING is returned, it is not defined which reason code is returned if other warning conditions exist.

The ability to specify and have accepted report options that are not recognized by the local queue manager is useful when sending a message with a report option that is recognized and processed by a *remote* queue manager.

Analyzing the report field:

The *Report* field contains subfields; because of this, applications that need to check whether the sender of the message requested a particular report must use one of the techniques described.

Using bit operations

If the programming language supports bit operations, perform the following steps:

1. Select one of the following values, according to the type of report to be checked:
   - MQRO_COA_WITH_FULL_DATA for COA report
   - MQRO_COD_WITH_FULL_DATA for COD report
   - MQRO_EXCEPTION_WITH_FULL_DATA for exception report
   - MQRO_EXPIRATION_WITH_FULL_DATA for expiration report

   Call the value `A`.

   On z/OS, use the MQRO_*_WITH_DATA values instead of the MQRO_*_WITH_FULL_DATA values.
2. Combine the *Report* field with `A` using the bitwise AND operation; call the result `B`.
3. Test `B` for equality with each value that is possible for that type of report.

   For example, if `A` is MQRO_EXCEPTION_WITH_FULL_DATA, test `B` for equality with each of the following to determine what was specified by the sender of the message:
   - MQRO_NONE
   - MQRO_EXCEPTION
   - MQRO_EXCEPTION_WITH_DATA
   - MQRO_EXCEPTION_WITH_FULL_DATA

   The tests can be performed in whatever order is most convenient for the application logic.

Use a similar method to test for the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options; select as the value `A` whichever of these two constants is appropriate, and then proceed as described above.

Using arithmetic

If the programming language *does not* support bit operations, perform the following steps using integer arithmetic:

1. Select one of the following values, according to the type of report to be checked:

- MQRO_COA for COA report
- MQRO_COD for COD report
- MQRO_EXCEPTION for exception report
- MQRO_EXPIRATION for expiration report

Call the value A.

2. Divide the *Report* field by A; call the result B.
3. Divide B by 8; call the result C.
4. Multiply C by 8 and subtract from B; call the result D.
5. Multiply D by A; call the result E.
6. Test E for equality with each value that is possible for that type of report.

   For example, if A is MQRO_EXCEPTION, test E for equality with each of the following to determine what was specified by the sender of the message:

   - MQRO_NONE
   - MQRO_EXCEPTION
   - MQRO_EXCEPTION_WITH_DATA
   - MQRO_EXCEPTION_WITH_FULL_DATA

   The tests can be performed in whatever order is most convenient for the application logic.

The following pseudocode illustrates this technique for exception report messages:

```
A = MQRO_EXCEPTION
B = Report/A
C = B/8
D = B - C*8
E = D*A
```

Use a similar method to test for the MQRO_PASS_MSG_ID or MQRO_PASS_CORREL_ID options; select as the value A whichever of these two constants is appropriate, and then proceed as described above, but replacing the value 8 in the steps above by the value 2.

Structure of the message-flags field:

This information describes the structure of the message-flags field.

The *MsgFlags* field is a 32-bit integer that is divided into three separate subfields. These subfields identify:

- Message flags that are rejected if the local queue manager does not recognize them
- Message flags that are always accepted, even if the local queue manager does not recognize them
- Message flags that are accepted only if certain other conditions are satisfied

Note: All subfields in *MsgFlags* are reserved for use by the queue manager.

Each subfield is identified by a bit mask that has 1-bits in the positions corresponding to the subfield, and 0-bits elsewhere. The bits are numbered such that bit 0 is the most significant bit, and bit 31 the least significant bit. The following masks are defined to identify the subfields:

MQMF_REJECT_UNSUP_MASK

This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_MSG_FLAGS_ERROR.

This subfield occupies bit positions 20 through 31.

The following message flags are included in this subfield:

- MQMF_LAST_MSG_IN_GROUP

- MQMF_LAST_SEGMENT
- MQMF_MSG_IN_GROUP
- MQMF_SEGMENT
- MQMF_SEGMENTATION_ALLOWED
- MQMF_SEGMENTATION_INHIBITED

MQMF_ACCEPT_UNSUP_MASK
This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls. The completion code is MQCC_OK.

This subfield occupies bit positions 0 through 11.

MQMF_ACCEPT_UNSUP_IF_XMIT_MASK
This mask identifies the bit positions within the *MsgFlags* field where message flags that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls *provided* that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue (that is, the queue identified by the *ObjectQMgrName* and *ObjectName* fields in the object descriptor specified on the MQOPEN or MQPUT1 call is not a local transmission queue).

Completion code MQCC_OK is returned if these conditions are satisfied, and MQCC_FAILED with reason code MQRC_MSG_FLAGS_ERROR if not.

This subfield occupies bit positions 12 through 19.

If there are flags specified in the *MsgFlags* field that the queue manager does not recognize, the queue manager checks each subfield in turn by using the bitwise AND operation to combine the *MsgFlags* field with the mask for that subfield. If the result of that operation is not zero, the completion code and reason codes described above are returned.

Data conversion

This collection of topics describes the interface to the data-conversion exit, and the processing performed by the queue manager when data conversion is required.

For more information about data conversion, see the document *Data Conversion under WebSphere MQ* at http://www.ibm.com/support/docview.wss?uid=swg27005729.

The data-conversion exit is invoked as part of the processing of the MQGET call in order to convert the application message data to the representation required by the receiving application. Conversion of the application message data is optional; it requires the MQGMO_CONVERT option to be specified on the MQGET call.

The following subjects are described:

- The processing performed by the queue manager in response to the MQGMO_CONVERT option; see "Conversion processing" on page 2211.
- Processing conventions used by the queue manager when processing a built-in format; these conventions are recommended for user-written exits too. See "Processing conventions" on page 2212.
- Special considerations for converting report messages; see "Conversion of report messages" on page 2216.
- The parameters passed to the data-conversion exit; see "MQ_DATA_CONV_EXIT - Data conversion exit" on page 2229.
- A call that can be used from the exit to convert character data between different representations; see "MQXCNVC - Convert characters" on page 2223.

- The data-structure parameter that is specific to the exit; see "MQDXP - Data-conversion exit parameter" on page 2216.

Conversion processing:

This information describes the processing performed by the queue manager in response to the MQGMO_CONVERT option.

The queue manager performs the following actions if the MQGMO_CONVERT option is specified on the MQGET call, and there is a message to be returned to the application:

1. If one or more of the following is true, no conversion is necessary:
   - The message data is already in the character set and encoding required by the application issuing the MQGET call. The application must set the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter of the MQGET call to the values required, before issuing the call.
   - The length of the message data is zero.
   - The length of the *Buffer* parameter of the MQGET call is zero.

   In these cases the message is returned without conversion to the application issuing the MQGET call; the *CodedCharSetId* and *Encoding* values in the *MsgDesc* parameter are set to the values in the control information in the message, and the call completes with one of the following combinations of completion code and reason code:

Completion code	Reason code
MQCC_OK	MQRC_NONE
MQCC_WARNING	MQRC_TRUNCATED_MSG_ACCEPTED
MQCC_WARNING	MQRC_TRUNCATED_MSG_FAILED

   The following steps are performed only if the character set or encoding of the message data differs from the corresponding value in the *MsgDesc* parameter, and there is data to be converted:
2. If the *Format* field in the control information in the message has the value MQFMT_NONE, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

   In all other cases conversion processing continues.
3. The message is removed from the queue and placed in a temporary buffer that is the same size as the *Buffer* parameter. For browse operations, the message is copied into the temporary buffer, instead of being removed from the queue.
4. If the message has to be truncated to fit in the buffer, the following is done:
   - If the MQGMO_ACCEPT_TRUNCATED_MSG option was *not* specified, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_TRUNCATED_MSG_FAILED.
   - If the MQGMO_ACCEPT_TRUNCATED_MSG option *was* specified, the completion code is set to MQCC_WARNING, the reason code is set to MQRC_TRUNCATED_MSG_ACCEPTED, and conversion processing continues.
5. If the message can be accommodated in the buffer without truncation, or the MQGMO_ACCEPT_TRUNCATED_MSG option was specified, the following is done:
   - If the format is a built-in format, the buffer is passed to the queue-manager's data-conversion service.
   - If the format is not a built-in format, the buffer is passed to a user-written exit with the same name as the format. If the exit cannot be found, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_FORMAT_ERROR.

   If no error occurs, the output from the data-conversion service or from the user-written exit is the converted message, plus the completion code and reason code to be returned to the application issuing the MQGET call.

6. If the conversion is successful, the queue manager returns the converted message to the application. In this case, the completion code and reason code returned by the MQGET call are one of the following combinations:

Completion code	Reason code
MQCC_OK	MQRC_NONE
MQCC_WARNING	MQRC_TRUNCATED_MSG_ACCEPTED

However, if the conversion is performed by a user-written exit, other reason codes can be returned, even when the conversion is successful.

If the conversion fails, the queue manager returns the unconverted message to the application, with the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter set to the values in the control information in the message, and with completion code MQCC_WARNING.

Processing conventions:

When converting a built-in format, the queue manager follows the processing conventions described.

User-written exits should also follow these conventions, although this is not enforced by the queue manager. The built-in formats converted by the queue manager are:

- MQFMT_ADMIN
- MQFMT_CICS (z/OS only)
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_DIST_HEADER
- MQFMT_EVENT version 1
- MQFMT_EVENT version 2
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_MD_EXTENSION
- MQFMT_PCF
- MQFMT_REF_MSG_HEADER
- MQFMT_RF_HEADER
- MQFMT_RF_HEADER_2
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_WORK_INFO_HEADER (z/OS only)
- MQFMT_XMIT_Q_HEADER

1. If the message expands during conversion, and exceeds the size of the *Buffer* parameter, the following is done:
   - If the MQGMO_ACCEPT_TRUNCATED_MSG option was *not* specified, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG.
   - If the MQGMO_ACCEPT_TRUNCATED_MSG option *was* specified, the message is truncated, the completion code is set to MQCC_WARNING, the reason code is set to MQRC_TRUNCATED_MSG_ACCEPTED, and conversion processing continues.
2. If truncation occurs (either before or during conversion), the number of valid bytes returned in the *Buffer* parameter can be *less than* the length of the buffer.

This can occur, for example, if a 4-byte integer or a DBCS character straddles the end of the buffer. The incomplete element of information is not converted, and those bytes in the returned message do not contain valid information. This can also occur if a message that was truncated before conversion shrinks during conversion.

If the number of valid bytes returned is less than the length of the buffer, the unused bytes at the end of the buffer are set to nulls.

3. If an array or string straddles the end of the buffer, as much of the data as possible is converted; only the particular array element or DBCS character which is incomplete is not converted; preceding array elements or characters are converted.
4. If truncation occurs (either before or during conversion), the length returned for the *DataLength* parameter is the length of the *unconverted* message before truncation.
5. When strings are converted between single-byte character sets (SBCS), double-byte character sets (DBCS), or multi-byte character sets (MBCS), the strings can expand or contract.
   - In the PCF formats MQFMT_ADMIN, MQFMT_EVENT, and MQFMT_PCF, the strings in the MQCFST and MQCFSL structures expand or contract as necessary to accommodate the string after conversion.

     For the string-list structure MQCFSL, the strings in the list might expand or contract by different amounts. If this happens, the queue manager pads the shorter strings with blanks to make them the same length as the longest string after conversion.
   - In the format MQFMT_REF_MSG_HEADER, the strings addressed by the *SrcEnvOffset*, *SrcNameOffset*, *DestEnvOffset*, and *DestNameOffset* fields expand or contract as necessary to accommodate the strings after conversion.
   - In the format MQFMT_RF_HEADER, the *NameValueString* field expands or contracts as necessary to accommodate the name/value pairs after conversion.
   - In structures with fixed field sizes, the queue manager allows strings to expand or contract within their fixed fields, provided that no significant information is lost. In this regard, trailing blanks and characters following the first null character in the field are treated as insignificant.
     – If the string expands, but only insignificant characters need to be discarded to accommodate the converted string in the field, the conversion succeeds and the call completes with MQCC_OK and reason code MQRC_NONE (assuming no other errors).
     – If the string expands, but the converted string requires significant characters to be discarded in order to fit in the field, the message is returned unconverted and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_STRING_TOO_BIG.

       Note: Reason code MQRC_CONVERTED_STRING_TOO_BIG results in this case whether or not the MQGMO_ACCEPT_TRUNCATED_MSG option was specified.
     – If the string contracts, the queue manager pads the string with blanks to the length of the field.
6. For messages consisting of one or more MQ header structures followed by user data, one or more of the header structures might be converted, while the remainder of the message is not. However, (with two exceptions) the *CodedCharSetId* and *Encoding* fields in each header structure always correctly indicate the character set and encoding of the data that follows the header structure.

   The two exceptions are the MQCIH and MQIIH structures, where the values in the *CodedCharSetId* and *Encoding* fields in those structures are not significant. For those structures, the data following the structure is in the same character set and encoding as the MQCIH or MQIIH structure itself.
7. If the *CodedCharSetId* or *Encoding* fields in the control information of the message being retrieved, or in the *MsgDesc* parameter, specify values that are undefined or not supported, the queue manager might ignore the error if the undefined or unsupported value does not need to be used in converting the message.

   For example, if the *Encoding* field in the message specifies an unsupported float encoding, but the message contains only integer data, or contains floating-point data that does not require conversion (because the source and target float encodings are identical), the error might not be diagnosed.

If the error is diagnosed, the message is returned unconverted, with completion code MQCC_WARNING and one of the MQRC_SOURCE_*_ERROR or MQRC_TARGET_*_ERROR reason codes (as appropriate); the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to the values in the control information in the message.

If the error is not diagnosed and the conversion completes successfully, the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are those specified by the application issuing the MQGET call.

8. In all cases, if the message is returned to the application unconverted the completion code is set to MQCC_WARNING, and the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter are set to the values appropriate to the unconverted data. This is done for MQFMT_NONE also.

   The *Reason* parameter is set to a code that indicates why the conversion could not be carried out, unless the message also had to be truncated; reason codes related to truncation take precedence over reason codes related to conversion. (To determine if a truncated message was converted, check the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter.)

   When an error is diagnosed, either a specific reason code is returned, or the general reason code MQRC_NOT_CONVERTED. The reason code returned depends on the diagnostic capabilities of the underlying data-conversion service.

9. If completion code MQCC_WARNING is returned, and more than one reason code is relevant, the order of precedence is as follows:
   a. The following reasons take precedence over all others; only one of the reasons in this group can arise:
      - MQRC_SIGNAL_REQUEST_ACCEPTED
      - MQRC_TRUNCATED_MSG_ACCEPTED
   b. The order of precedence within the remaining reason codes is not defined.

10. On completion of the MQGET call:
    - The following reason code indicates that the message was converted successfully:
      – MQRC_NONE
    - The following reason codes indicate that the message *might* have been converted successfully (check the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to find out):
      – MQRC_MSG_MARKED_BROWSE_CO_OP
      – MQRC_TRUNCATED_MSG_ACCEPTED
    - All other reason codes indicate that the message was not converted.

    The following processing is specific to the built-in formats; it does not apply to user-defined formats:

11. With the exception of the following formats:
    - MQFMT_ADMIN
    - MQFMT_COMMAND_1
    - MQFMT_COMMAND_2
    - MQFMT_EVENT
    - MQFMT_IMS_VAR_STRING
    - MQFMT_PCF
    - MQFMT_STRING

    none of the built-in formats can be converted from or to character sets that do not have SBCS characters for the characters that are valid in queue names. If an attempt is made to perform such a conversion, the message is returned unconverted, with completion code MQCC_WARNING and reason code MQRC_SOURCE_CCSID_ERROR or MQRC_TARGET_CCSID_ERROR, as appropriate.

    The Unicode character set UCS-2 is an example of a character set that does not have SBCS characters for the characters that are valid in queue names.

12. If the message data for a built-in format is truncated, fields within the message that contain lengths of strings, or counts of elements or structures, are *not* adjusted to reflect the length of the data

actually returned to the application; the values returned for such fields within the message data are the values applicable to the message *before truncation*.

When processing messages such as a truncated MQFMT_ADMIN message, ensure that the application does not attempt to access data beyond the end of the data returned.

13. If the format name is MQFMT_DEAD_LETTER_HEADER, the message data begins with an MQDLH structure, possibly followed by zero or more bytes of application message data. The format, character set, and encoding of the application message data are defined by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQDLH structure at the start of the message. Because the MQDLH structure and application message data can have different character sets and encodings, one, other, or both of the MQDLH structure and application message data might require conversion.

    The queue manager converts the MQDLH structure first, as necessary. If conversion is successful, or the MQDLH structure does not require conversion, the queue manager checks the *CodedCharSetId* and *Encoding* fields in the MQDLH structure to see if conversion of the application message data is required. If conversion *is* required, the queue manager invokes the user-written exit with the name given by the *Format* field in the MQDLH structure, or performs the conversion itself (if *Format* is the name of a built-in format).

    If the MQGET call returns a completion code of MQCC_WARNING, and the reason code is one of those indicating that conversion was not successful, one of the following applies:

    - The MQDLH structure could not be converted. In this case the application message data will not have been converted either.
    - The MQDLH structure was converted, but the application message data was not.

    The application can examine the values returned in the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter, and those in the MQDLH structure, in order to determine which of the above applies.

14. If the format name is MQFMT_XMIT_Q_HEADER, the message data begins with an MQXQH structure, possibly followed by zero or more bytes of additional data. This additional data is usually the application message data (which may be of zero length), but there can also be one or more further MQ header structures present, at the start of the additional data.

    The MQXQH structure must be in the character set and encoding of the queue manager. The format, character set, and encoding of the data following the MQXQH structure are given by the *Format*, *CodedCharSetId*, and *Encoding* fields in the MQMD structure contained *within* the MQXQH. For each subsequent MQ header structure present, the *Format*, *CodedCharSetId*, and *Encoding* fields in the structure describe the data that follows that structure; that data is either another MQ header structure, or the application message data.

    If the MQGMO_CONVERT option is specified for an MQFMT_XMIT_Q_HEADER message, the application message data and certain of the MQ header structures are converted, *but the data in the MQXQH structure is not*. On return from the MQGET call, therefore:

    - The values of the *Format*, *CodedCharSetId*, and *Encoding* fields in the *MsgDesc* parameter describe the data in the MQXQH structure, and *not* the application message data; the values are therefore *not* the same as those specified by the application that issued the MQGET call.

      The effect of this is that an application that repeatedly gets messages from a transmission queue with the MQGMO_CONVERT option specified must reset the *CodedCharSetId* and *Encoding* fields in the *MsgDesc* parameter to the values required for the application message data, before each MQGET call.
    - The values of the *Format*, *CodedCharSetId*, and *Encoding* fields in the last MQ header structure present describe the application message data. If there are no other MQ header structures present, the application message data is described by these fields in the MQMD structure within the MQXQH structure. If conversion is successful, the values will be the same as those specified in the *MsgDesc* parameter by the application that issued the MQGET call.

    If the message is a distribution-list message, the MQXQH structure is followed by an MQDH structure (plus its arrays of MQOR and MQPMR records), which in turn might be followed by zero or more further MQ header structures and zero or more bytes of application message data. Like the

MQXQH structure, the MQDH structure must be in the character set and encoding of the queue manager, and it is not converted on the MQGET call, even if the MQGMO_CONVERT option is specified.

The processing of the MQXQH and MQDH structures described above is primarily intended for use by message channel agents when they get messages from transmission queues.

Conversion of report messages:

In general a report message can contain varying amounts of application message data, according to the report options specified by the sender of the original message. However, an activity report can contain data but without the report option mentioning *_WITH_DATA in the constant.

In particular, a report message can contain either:

1. No application message data
2. Some of the application message data from the original message

   This occurs when the sender of the original message specifies MQRO_*_WITH_DATA and the message is longer than 100 bytes.
3. All the application message data from the original message

   This occurs when the sender of the original message specifies MQRO_*_WITH_FULL_DATA, or specifies MQRO_*_WITH_DATA and the message is 100 bytes or shorter.

When the queue manager or message channel agent generates a report message, it copies the format name from the original message into the *Format* field in the control information in the report message. The format name in the report message might therefore imply a length of data that is different from the length actually present in the report message (cases 1 and 2 above).

If the MQGMO_CONVERT option is specified when the report message is retrieved:

- For case 1 above, the data-conversion exit is not invoked (because the report message has no data).
- For case 3 above, the format name correctly implies the length of the message data.
- But for case 2 above, the data-conversion exit is invoked to convert a message that is *shorter* than the length implied by the format name.

  In addition, the reason code passed to the exit is usually MQRC_NONE (that is, the reason code does not indicate that the message has been truncated). This happens because the message data was truncated by the *sender* of the report message, and not by the receiver's queue manager in response to the MQGET call.

Because of these possibilities, the data-conversion exit must *not* use the format name to deduce the length of data passed to it; instead the exit must check the length of data provided, and be prepared to convert *less* data than the length implied by the format name. If the data can be converted successfully, completion code MQCC_OK and reason code MQRC_NONE must be returned by the exit. The length of the message data to be converted is passed to the exit as the *InBufferLength* parameter.

Product-sensitive programming interface

MQDXP - Data-conversion exit parameter:

The MQDXP structure is a parameter that the queue manager passes to the data-conversion exit when the exit is invoked to convert the message data as part of the processing of the MQGET call. See the description of the MQ_DATA_CONV_EXIT call for details of the data conversion exit.

Character data in MQDXP is in the character set of the local queue manager; this is given by the *CodedCharSetId* queue-manager attribute. Numeric data in MQDXP is in the native machine encoding; this is given by MQENC_NATIVE.

Only the *DataLength*, *CompCode*, *Reason*, and *ExitResponse* fields in MQDXP can be changed by the exit; changes to other fields are ignored. However, the *DataLength* field *cannot* be changed if the message being converted is a segment that contains only part of a logical message.

When control returns to the queue manager from the exit, the queue manager checks the values returned in MQDXP. If the values returned are not valid, the queue manager continues processing as though the exit had returned MQXDR_CONVERSION_FAILED in *ExitResponse*; however, the queue manager ignores the values of the *CompCode* and *Reason* fields returned by the exit in this case, and uses instead the values those fields had on *input* to the exit. The following values in MQDXP cause this processing to occur:

- *ExitResponse* field not MQXDR_OK and not MQXDR_CONVERSION_FAILED
- *CompCode* field not MQCC_OK and not MQCC_WARNING
- *DataLength* field less than zero, or *DataLength* field changed when the message being converted is a segment that contains only part of a logical message.

The following table summarizes the fields in the structure.

Table 226. Fields in MQDXP

Field	Description	Topic
StrucId	Structure identifier	StrucId
Version	Structure version number	Version
AppOptions	Application options	AppOptions
Encoding	Numeric encoding required by application	Encoding
CodedCharSetId	Character set required by application	CodedCharSetId
DataLength	Length in bytes of message data	DataLength
CompCode	Completion code	CompCode
Reason	Reason code qualifying *CompCode*	Reason
ExitResponse	Response from exit	ExitResponse
Hconn	Connection handle	Hconn
pEntryPoints	Address of the MQIEP structure	pEntryPoints

Fields

The MQDXP structure contains the following fields; the fields are described in alphabetical order.

AppOptions
Type: MQLONG

This is a copy of the *Options* field of the MQGMO structure specified by the application issuing the MQGET call. The exit might need to examine these to ascertain whether the MQGMO_ACCEPT_TRUNCATED_MSG option was specified.

This is an input field to the exit.

CodedCharSetId
Type: MQLONG

This is the coded character-set identifier of the character set required by the application issuing the MQGET call; see the *CodedCharSetId* field in the MQMD structure for more details. If the application

specifies the special value MQCCSI_Q_MGR on the MQGET call, the queue manager changes this to the actual character-set identifier of the character set used by the queue manager, before invoking the exit.

If the conversion is successful, the exit must copy this to the *CodedCharSetId* field in the message descriptor.

This is an input field to the exit.

CompCode
Type: MQLONG

When the exit is invoked, this contains the completion code that is returned to the application that issued the MQGET call, if the exit does nothing. It is always MQCC_WARNING, because either the message was truncated, or the message requires conversion and this has not yet been done.

On output from the exit, this field contains the completion code to be returned to the application in the *CompCode* parameter of the MQGET call; only MQCC_OK and MQCC_WARNING are valid. See the description of the *Reason* field for suggestions on how the exit can set this field on output.

This is an input/output field to the exit.

DataLength
Type: MQLONG

When the exit is invoked, this field contains the original length of the application message data. If the message was truncated to fit into the buffer provided by the application, the size of the message provided to the exit is *smaller* than the value of *DataLength*. The size of the message provided to the exit is always given by the *InBufferLength* parameter of the exit, irrespective of any truncation that has occurred.

Truncation is indicated by the *Reason* field having the value MQRC_TRUNCATED_MSG_ACCEPTED on input to the exit.

Most conversions do not need to change this length, but an exit can do so if necessary; the value set by the exit is returned to the application in the *DataLength* parameter of the MQGET call. However, this length *cannot* be changed if the message being converted is a segment that contains only part of a logical message. This is because changing the length would cause the offsets of later segments in the logical message to be incorrect.

Note that, if the exit wants to change the length of the data, be aware that the queue manager has already decided whether the message data fits into the application's buffer, based on the length of the *unconverted* data. This decision determines whether the message is removed from the queue (or the browse cursor moved, for a browse request), and is not affected by any change to the data length caused by the conversion. For this reason it is recommended that conversion exits do not cause a change in the length of the application message data.

If character conversion does imply a change of length, a string can be converted into another string with the same length in bytes, truncating trailing blanks, or padding with blanks as necessary.

The exit is not invoked if the message contains no application message data; hence *DataLength* is always greater then zero.

This is an input/output field to the exit.

Encoding
Type: MQLONG

Numeric encoding required by application.

This is the numeric encoding required by the application issuing the MQGET call; see the *Encoding* field in the MQMD structure for more details.

If the conversion is successful, the exit copies this to the *Encoding* field in the message descriptor.

This is an input field to the exit.

ExitOptions
Type: MQLONG

This is a reserved field; its value is `0`.

ExitResponse
Type: MQLONG

Response from exit. This is set by the exit to indicate the success or otherwise of the conversion. It must be one of the following:

MQXDR_OK
Conversion was successful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *CompCode* field on output from the exit
- The value of the *Reason* field on output from the exit
- The value of the *DataLength* field on output from the exit
- The contents of the exit's output buffer *OutBuffer*. The number of bytes returned is the lesser of the exit's *OutBufferLength* parameter, and the value of the *DataLength* field on output from the exit.

  If the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter are *both* unchanged, the queue manager returns:
- The value of the *Encoding* and *CodedCharSetId* fields in the MQDXP structure on *input* to the exit.

  If one or both of the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter has been changed, the queue manager returns:
- The value of the *Encoding* and *CodedCharSetId* fields in the exit's message descriptor parameter on output from the exit

MQXDR_CONVERSION_FAILED
Conversion was unsuccessful.

If the exit specifies this value, the queue manager returns the following to the application that issued the MQGET call:

- The value of the *CompCode* field on output from the exit
- The value of the *Reason* field on output from the exit
- The value of the *DataLength* field on *input* to the exit
- The contents of the exit's input buffer *InBuffer*. The number of bytes returned is given by the *InBufferLength* parameter

If the exit has altered *InBuffer*, the results are undefined.

ExitResponse is an output field from the exit.

Hconn
Type: MQHCONN

This is a connection handle which can be used on the MQXCNVC call. This handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

pEntryPoints
Type: PMQIEP

The address of an MQIEP structure through which MQI and DCI calls can be made.

Reason

Type: MQLONG

Reason code qualifying *CompCode*.

When the exit is invoked, this contains the reason code that is returned to the application that issued the MQGET call, if the exit chooses to do nothing. Among possible values are MQRC_TRUNCATED_MSG_ACCEPTED, indicating that the message was truncated in order fit into the buffer provided by the application, and MQRC_NOT_CONVERTED, indicating that the message requires conversion but that this has not yet been done.

On output from the exit, this field contains the reason to be returned to the application in the *Reason* parameter of the MQGET call; the following is recommended:

- If *Reason* had the value MQRC_TRUNCATED_MSG_ACCEPTED on input to the exit, the *Reason* and *CompCode* fields must not be altered, irrespective of whether the conversion succeeds or fails.

  (If the *CompCode* field is not MQCC_OK, the application which retrieves the message can identify a conversion failure by comparing the returned *Encoding* and *CodedCharSetId* values in the message descriptor with the values requested; in contrast, the application cannot distinguish a truncated message from a message that fitted the buffer. For this reason, MQRC_TRUNCATED_MSG_ACCEPTED must be returned in preference to any of the reasons that indicate conversion failure.)
- If *Reason* had any other value on input to the exit:
  - If the conversion succeeds, *CompCode* must be set to MQCC_OK and *Reason* set to MQRC_NONE.
  - If the conversion fails, or the message expands and has to be truncated to fit in the buffer, *CompCode* must be set to MQCC_WARNING (or left unchanged), and *Reason* set to one of the values listed, to indicate the nature of the failure.

    Note if the message after conversion is too large for the buffer, it must be truncated only if the application that issued the MQGET call specified the MQGMO_ACCEPT_TRUNCATED_MSG option:
    - If it did specify that option, reason MQRC_TRUNCATED_MSG_ACCEPTED is returned.
    - If it did not specify that option, the message is returned unconverted, with reason code MQRC_CONVERTED_MSG_TOO_BIG.

The reason codes listed are recommended for use by the exit to indicate the reason that conversion failed, but the exit can return other values from the set of MQRC_* codes if deemed appropriate. In addition, the range of values MQRC_APPL_FIRST through MQRC_APPL_LAST are allocated for use by the exit to indicate conditions that the exit wants to communicate to the application issuing the MQGET call.

Note: If the message cannot be converted successfully, the exit *must* return MQXDR_CONVERSION_FAILED in the *ExitResponse* field, in order to cause the queue manager to return the unconverted message. This is true regardless of the reason code returned in the *Reason* field.

MQRC_APPL_FIRST
(900, X'384') Lowest value for application-defined reason code.

MQRC_APPL_LAST
(999, X'3E7') Highest value for application-defined reason code.

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too large for buffer.

MQRC_NOT_CONVERTED
(2119, X'847') Message data not converted.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR
(2113, X'841') Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR
(2114, X'842') Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'846') Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).

This is an input/output field to the exit.

StrucId
Type: MQCHAR4

Structure identifier. The value must be:

MQDXP_STRUC_ID
Identifier for data conversion exit parameter structure.

For the C programming language, the constant MQDXP_STRUC_ID_ARRAY is also defined; this has the same value as MQDXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version
Type: MQLONG

Structure version number. The value must be:

MQDXP_VERSION_1
Version number for data-conversion exit parameter structure.

The following constant specifies the version number of the current version:

MQDXP_CURRENT_VERSION
Current version of data-conversion exit parameter structure.

Note: When a new version of this structure is introduced, the layout of the existing part is not changed. The exit must therefore check that the *Version* field is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

C declaration

```
typedef struct tagMQDXP MQDXP;
struct tagMQDXP {
  MQCHAR4  StrucId;         /* Structure identifier */
```

```
  MQLONG   Version;         /* Structure version number */
  MQLONG   ExitOptions;     /* Reserved */
  MQLONG   AppOptions;      /* Application options */
  MQLONG   Encoding;        /* Numeric encoding required by
                               application */
  MQLONG   CodedCharSetId;  /* Character set required by application */
  MQLONG   DataLength;      /* Length in bytes of message data */
  MQLONG   CompCode;        /* Completion code */
  MQLONG   Reason;          /* Reason code qualifying CompCode */
  MQLONG   ExitResponse;    /* Response from exit */
  MQHCONN  Hconn;           /* Connection handle */
  PMQIEP   pEntryPoints;    /* Address of the MQIEP structure */
};
```

COBOL declaration (IBM i only)

```
**   MQDXP structure
  10 MQDXP.
**    Structure identifier
   15 MQDXP-STRUCID        PIC X(4).
**    Structure version number
   15 MQDXP-VERSION        PIC S9(9) BINARY.
**    Reserved
   15 MQDXP-EXITOPTIONS    PIC S9(9) BINARY.
**    Application options
   15 MQDXP-APPOPTIONS     PIC S9(9) BINARY.
**    Numeric encoding required by application
   15 MQDXP-ENCODING       PIC S9(9) BINARY.
**    Character set required by application
   15 MQDXP-CODEDCHARSETID PIC S9(9) BINARY.
**    Length in bytes of message data
   15 MQDXP-DATALENGTH     PIC S9(9) BINARY.
**    Completion code
   15 MQDXP-COMPCODE       PIC S9(9) BINARY.
**    Reason code qualifying COMPCODE
   15 MQDXP-REASON         PIC S9(9) BINARY.
**    Response from exit
   15 MQDXP-EXITRESPONSE   PIC S9(9) BINARY.
**    Connection handle
   15 MQDXP-HCONN          PIC S9(9) BINARY.
```

System/390 assembler declaration

```
MQDXP                 DSECT
MQDXP_STRUCID         DS   CL4  Structure identifier
MQDXP_VERSION         DS   F    Structure version number
MQDXP_EXITOPTIONS     DS   F    Reserved
MQDXP_APPOPTIONS      DS   F    Application options
MQDXP_ENCODING        DS   F    Numeric encoding required by application
MQDXP_CODEDCHARSETID  DS   F    Character set required by application
MQDXP_DATALENGTH      DS   F    Length in bytes of message data
MQDXP_COMPCODE        DS   F    Completion code
MQDXP_REASON          DS   F    Reason code qualifying COMPCODE
MQDXP_EXITRESPONSE    DS   F    Response from exit
MQDXP_HCONN           DS   F    Connection handle
*
MQDXP_LENGTH          EQU  *-MQDXP
                      ORG  MQDXP
MQDXP_AREA            DS   CL(MQDXP_LENGTH)
```

MQXCNVC - Convert characters:

The MQXCNVC call converts characters from one character set to another using the C programming language.

This call is part of the WebSphere MQ Data Conversion Interface (DCI), which is one of the WebSphere MQ framework interfaces.

Note: The call can be used from both application, and data-conversion exit environments.

Syntax

MQXCNVC (*Hconn*, *Options*, *SourceCCSID*, *SourceLength*, *SourceBuffer*, *TargetCCSID*, *TargetLength*, *TargetBuffer*, *DataLength*, *CompCode*, *Reason*)

Parameters

Hconn

Type: MQHCONN - input

This handle represents the connection to the queue manager.

In a data-conversion exit, *Hconn* is normally the handle that is passed to the data-conversion exit in the *Hconn* field of the MQDXP structure; this handle is not necessarily the same as the handle specified by the application which issued the MQGET call.

On IBM i, the following special value can be specified for *Hconn*:

MQHC_DEF_HCONN
Default connection handle.

If you run a CICS TS 3.2 or higher application, ensure that the character conversion exit program, which invokes the MQXCNVC call, is defined as OPENAPI. This definition prevents the 2018 MQRC_HCONN_ERROR error caused by from an incorrect connection, and allows the MQGET to complete.

Options

Type: MQLONG - input

Options that control the action of MQXCNVC.

Zero or more of the options described can be specified. If more than one is required, the values can be:

- Added (do not add the same constant more than once), or
- Combined using the bitwise OR operation (if the programming language supports bit operations)

Default-conversion option: The following option controls the use of default character conversion:

MQDCC_DEFAULT_CONVERSION
Default conversion.

This option specifies that default character conversion can be used if one or both of the character sets specified on the call is not supported. This allows the queue manager to use an installation-specified default character set that approximates the specified character set, when converting the string.

Note: The result of using an approximate character set to convert the string is that some characters can be converted incorrectly. This can be avoided by using in the string only characters which are common to both the specified character set and the default character set.

The default character sets are defined by a configuration option when the queue manager is installed or restarted.

If MQDCC_DEFAULT_CONVERSION is not specified, the queue manager uses only the specified character sets to convert the string, and the call fails if one or both of the character sets is not supported.

This option is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows.

Padding option: The following option allows the queue manager to pad the converted string with blanks or discard insignificant trailing characters, in order to make the converted string fit the target buffer:

MQDCC_FILL_TARGET_BUFFER
Fill target buffer.

This option requests that conversion take place in such a way that the target buffer is filled completely:

- If the string contracts when it is converted, trailing blanks are added in order to fill the target buffer.
- If the string expands when it is converted, trailing characters that are not significant are discarded to make the converted string fit the target buffer. If this can be done successfully, the call completes with MQCC_OK and reason code MQRC_NONE.

  If there are too few insignificant trailing characters, as much of the string as can fit is placed in the target buffer, and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG.

  Insignificant characters are:

  – Trailing blanks
  – Characters following the first null character in the string (but excluding the first null character itself)
- If the string, *TargetCCSID*, and *TargetLength* are such that the target buffer cannot be set completely with valid characters, the call fails with MQCC_FAILED and reason code MQRC_TARGET_LENGTH_ERROR. This can occur when *TargetCCSID* is a pure DBCS character set (such as UCS-2), but *TargetLength* specifies a length that is an odd number of bytes.
- *TargetLength* can be less than or greater than *SourceLength*. On return from MQXCNVC, *DataLength* has the same value as *TargetLength*.

If this option is not specified:

- The string is allowed to contract or expand within the target buffer as required. Insignificant trailing characters are not added or discarded.

  If the converted string fits in the target buffer, the call completes with MQCC_OK and reason code MQRC_NONE.

  If the converted string is too large for the target buffer, as much of the string as fits is placed in the target buffer, and the call completes with MQCC_WARNING and reason code MQRC_CONVERTED_MSG_TOO_BIG. Note fewer than *TargetLength* bytes can be returned in this case.
- *TargetLength* can be less than or greater than *SourceLength*. On return from MQXCNVC, *DataLength* is less than or equal to *TargetLength*.

This option is supported in the following environments: AIX, HP-UX, IBM i, Solaris, Linux, Windows.

Encoding options: The options described can be used to specify the integer encodings of the source and target strings. The relevant encoding is used *only* when the corresponding character set identifier indicates that the representation of the character set in main storage is dependent on the encoding used for binary integers. This affects only certain multibyte character sets (for example, UCS-2 character sets).

The encoding is ignored if the character set is a single-byte character set (SBCS), or a multibyte character set with representation in main storage that is not dependent on the integer encoding.

Only one of the MQDCC_SOURCE_* values must be specified, combined with one of the MQDCC_TARGET_* values:

MQDCC_SOURCE_ENC_NATIVE
Source encoding is the default for the environment and programming language.

MQDCC_SOURCE_ENC_NORMAL
Source encoding is normal.

MQDCC_SOURCE_ENC_REVERSED
Source encoding is reversed.

MQDCC_SOURCE_ENC_UNDEFINED
Source encoding is undefined.

MQDCC_TARGET_ENC_NATIVE
Target encoding is the default for the environment and programming language.

MQDCC_TARGET_ENC_NORMAL
Target encoding is normal.

MQDCC_TARGET_ENC_REVERSED
Target encoding is reversed.

MQDCC_TARGET_ENC_UNDEFINED
Target encoding is undefined.

The encoding values defined previously can be added directly to the *Options* field. However, if the source or target encoding is obtained from the *Encoding* field in the MQMD or other structure, the following processing must be done:

1. The integer encoding must be extracted from the *Encoding* field by eliminating the float and packed-decimal encodings; see "Analyzing encodings" on page 2205 for details of how to do this.
2. The integer encoding resulting from step 1 must be multiplied by the appropriate factor before being added to the *Options* field. These factors are:
   - MQDCC_SOURCE_ENC_FACTOR for the source encoding
   - MQDCC_TARGET_ENC_FACTOR for the target encoding

The following example code illustrates how this might be coded in the C programming language:

```
Options = (MsgDesc.Encoding & MQENC_INTEGER_MASK)
           * MQDCC_SOURCE_ENC_FACTOR
        + (DataConvExitParms.Encoding & MQENC_INTEGER_MASK)
           * MQDCC_TARGET_ENC_FACTOR;
```

If not specified, the encoding options default to undefined (MQDCC_*_ENC_UNDEFINED). In most cases, this does not affect the successful completion of the MQXCNVC call. However, if the corresponding character set is a multibyte character set with representation that is dependent on the encoding (for example, a UCS-2 character set), the call fails with reason code MQRC_SOURCE_INTEGER_ENC_ERROR or MQRC_TARGET_INTEGER_ENC_ERROR as appropriate.

The encoding options are supported in the following environments: AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows.

Default option: If none of the options described previously is specified, the following option can be used:

MQDCC_NONE
No options specified.

MQDCC_NONE is defined to aid program documentation. It is not intended that this option is used with any other, but as its value is zero, such use cannot be detected.

SourceCCSID
Type: MQLONG - input

This is the coded character set identifier of the input string in *SourceBuffer*.

SourceLength
Type: MQLONG - input

This is the length in bytes of the input string in *SourceBuffer*; it must be zero or greater.

SourceBuffer
Type: MQCHARxSourceLength - input

This is the buffer containing the string to be converted from one character set to another.

TargetCCSID
Type: MQLONG - input

This is the coded character set identifier of the character set to which *SourceBuffer* is to be converted.

TargetLength
Type: MQLONG - input

This is the length in bytes of the output buffer *TargetBuffer*; it must be zero or greater. It can be less than or greater than *SourceLength*.

TargetBuffer
Type: MQCHARxTargetLength - output

This is the string after it has been converted to the character set defined by *TargetCCSID*. The converted string can be shorter or longer than the unconverted string. The *DataLength* parameter indicates the number of valid bytes returned.

DataLength
Type: MQLONG - output

This is the length of the string returned in the output buffer *TargetBuffer*. The converted string can be shorter or longer than the unconverted string.

CompCode
Type: MQLONG - output

It is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted data too large for buffer.

If *CompCode* is MQCC_FAILED:

MQRC_DATA_LENGTH_ERROR
(2010, X'7DA') Data length parameter not valid.

MQRC_DBCS_ERROR
(2150, X'866') DBCS string not valid.

MQRC_HCONN_ERROR
(2018, X'7E2') Connection handle not valid.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_RESOURCE_PROBLEM
(2102, X'836') Insufficient system resources available.

MQRC_SOURCE_BUFFER_ERROR
(2145, X'861') Source buffer parameter not valid.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Source integer encoding not recognized.

MQRC_SOURCE_LENGTH_ERROR
(2143, X'85F') Source length parameter not valid.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_TARGET_BUFFER_ERROR
(2146, X'862') Target buffer parameter not valid.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Target integer encoding not recognized.

MQRC_TARGET_LENGTH_ERROR
(2144, X'860') Target length parameter not valid.

MQRC_UNEXPECTED_ERROR
(2195, X'893') Unexpected error occurred.

For detailed information about these codes, see Reason codes.

C invocation

```
MQXCNVC (Hconn, Options, SourceCCSID, SourceLength, SourceBuffer,
        TargetCCSID, TargetLength, TargetBuffer, &DataLength,
        &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;            /* Connection handle */
MQLONG   Options;          /* Options that control the action of
                              MQXCNVC */
MQLONG   SourceCCSID;      /* Coded character set identifier of string
                              before conversion */
MQLONG   SourceLength;     /* Length of string before conversion */
MQCHAR   SourceBuffer[n];  /* String to be converted */
MQLONG   TargetCCSID;      /* Coded character set identifier of string
```

```
                            after conversion */
MQLONG   TargetLength;     /* Length of output buffer */
MQCHAR   TargetBuffer[n];  /* String after conversion */
MQLONG   DataLength;       /* Length of output string */
MQLONG   CompCode;         /* Completion code */
MQLONG   Reason;           /* Reason code qualifying CompCode */
```

COBOL declaration (IBM i only)

```
     CALL 'MQXCNVC' USING HCONN, OPTIONS, SOURCECCSID, SOURCELENGTH,
                          SOURCEBUFFER, TARGETCCSID, TARGETLENGTH,
                          TARGETBUFFER, DATALENGTH, COMPCODE, REASON.
```

Declare the parameters as follows:

```
**   Connection handle
 01  HCONN         PIC S9(9) BINARY.
**   Options that control the action of MQXCNVC
 01  OPTIONS       PIC S9(9) BINARY.
**   Coded character set identifier of string before conversion
 01  SOURCECCSID   PIC S9(9) BINARY.
**   Length of string before conversion
 01  SOURCELENGTH  PIC S9(9) BINARY.
**   String to be converted
 01  SOURCEBUFFER  PIC X(n).
**   Coded character set identifier of string after conversion
 01  TARGETCCSID   PIC S9(9) BINARY.
**   Length of output buffer
 01  TARGETLENGTH  PIC S9(9) BINARY.
**   String after conversion
 01  TARGETBUFFER  PIC X(n).
**   Length of output string
 01  DATALENGTH    PIC S9(9) BINARY.
**   Completion code
 01  COMPCODE      PIC S9(9) BINARY.
**   Reason code qualifying COMPCODE
 01  REASON        PIC S9(9) BINARY.
```

S/390 assembler declaration

```
         CALL MQXCNVC,(HCONN,OPTIONS,SOURCECCSID,SOURCELENGTH,         X
               SOURCEBUFFER,TARGETCCSID,TARGETLENGTH,TARGETBUFFER,     X
               DATALENGTH,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN         DS  F      Connection handle
OPTIONS       DS  F      Options that control the action of MQXCNVC
SOURCECCSID   DS  F      Coded character set identifier of string before
*                        conversion
SOURCELENGTH  DS  F      Length of string before conversion
SOURCEBUFFER  DS  CL(n)  String to be converted
TARGETCCSID   DS  F      Coded character set identifier of string after
*                        conversion
TARGETLENGTH  DS  F      Length of output buffer
TARGETBUFFER  DS  CL(n)  String after conversion
DATALENGTH    DS  F      Length of output string
COMPCODE      DS  F      Completion code
REASON        DS  F      Reason code qualifying COMPCODE
```

MQ_DATA_CONV_EXIT - Data conversion exit:

The MQ_DATA_CONV_EXIT call describes the parameters that are passed to the data-conversion exit.

No entry point called MQ_DATA_CONV_EXIT is provided by the queue manager (see usage note 11).

This definition is part of the WebSphere MQ Data Conversion Interface (DCI), which is one of the WebSphere MQ framework interfaces.

Syntax

MQ_DATA_CONV_EXIT (*DataConvExitParms*, *MsgDesc*, *InBufferLength*, *InBuffer*, *OutBufferLength*, *OutBuffer*)

Parameters

DataConvExitParms
Type: MQDXP - input/output

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate the outcome of the conversion. See "MQDXP - Data-conversion exit parameter" on page 2216 for details of the fields in this structure.

MsgDesc
Type: MQMD - input/output

On input to the exit, this is the message descriptor associated with the message data passed to the exit in the *InBuffer* parameter.

Note: The *MsgDesc* parameter passed to the exit is always the most-recent version of MQMD supported by the queue manager which invokes the exit. If the exit is intended to be portable between different environments, the exit will check the *Version* field in *MsgDesc* to verify that the fields that the exit needs to access are present in the structure.

In the following environments, the exit is passed a version-2 MQMD: AIX, HP-UX, IBM i, Solaris, Linux, Windows. In all other environments that support the data conversion exit, the exit is passed a version-1 MQMD.

On output, the exit will change the *Encoding* and *CodedCharSetId* fields to the values requested by the application, if conversion was successful; these changes are reflected back to the application. Any other changes that the exit makes to the structure are ignored; they are not reflected back to the application.

If the exit returns MQXDR_OK in the *ExitResponse* field of the MQDXP structure, but does not change the *Encoding* or *CodedCharSetId* fields in the message descriptor, the queue manager returns for those fields the values that the corresponding fields in the MQDXP structure had on input to the exit.

InBufferLength
Type: MQLONG - input

Length in bytes of *InBuffer*.

This is the length of the input buffer *InBuffer*, and specifies the number of bytes to be processed by the exit. *InBufferLength* is the lesser of the length of the message data before conversion, and the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

InBuffer
Type: MQBYTExInBufferLength - input

Buffer containing the unconverted message.

This contains the message data before conversion. If the exit is unable to convert the data, the queue manager returns the contents of this buffer to the application after the exit has completed.

Note: The exit should not alter *InBuffer*; if this parameter is altered, the results are undefined.

In the C programming language, this parameter is defined as a pointer-to-void.

OutBufferLength
Type: MQLONG - input

Length in bytes of *OutBuffer*.

This is the length of the output buffer *OutBuffer*, and is the same as the length of the buffer provided by the application on the MQGET call.

The value is always greater than zero.

OutBuffer
Type: MQBYTExOutBufferLength - output

Buffer containing the converted message.

On output from the exit, if the conversion was successful (as indicated by the value MQXDR_OK in the *ExitResponse* field of the *DataConvExitParms* parameter), *OutBuffer* contains the message data to be delivered to the application, in the requested representation. If the conversion was unsuccessful, any changes that the exit has made to this buffer are ignored.

In the C programming language, this parameter is defined as a pointer-to-void.

Usage notes

1. A data-conversion exit is a user-written exit which receives control during the processing of an MQGET call. The function performed by the data-conversion exit is defined by the provider of the exit; however, the exit must conform to the rules described here, and in the associated parameter structure MQDXP.

   The programming languages that can be used for a data-conversion exit are determined by the environment.
2. The exit is invoked only if *all* of the following are true:
   - The MQGMO_CONVERT option is specified on the MQGET call
   - The *Format* field in the message descriptor is not MQFMT_NONE
   - The message is not already in the required representation; that is, one or both of the message's *CodedCharSetId* and *Encoding* is different from the value specified by the application in the message descriptor supplied on the MQGET call
   - The queue manager has not already done the conversion successfully
   - The length of the application's buffer is greater than zero
   - The length of the message data is greater than zero
   - The reason code so far during the MQGET operation is MQRC_NONE or MQRC_TRUNCATED_MSG_ACCEPTED
3. When an exit is being written, consider coding the exit in a way that allows it to convert messages that have been truncated. Truncated messages can arise in the following ways:
   - The receiving application provides a buffer that is smaller than the message, but specifies the MQGMO_ACCEPT_TRUNCATED_MSG option on the MQGET call.

     In this case, the *Reason* field in the *DataConvExitParms* parameter on input to the exit has the value MQRC_TRUNCATED_MSG_ACCEPTED.
   - The sender of the message truncated it before sending it. This can happen with report messages, for example (see "Conversion of report messages" on page 2216 for more details).

In this case, the *Reason* field in the *DataConvExitParms* parameter on input to the exit has the value MQRC_NONE (if the receiving application provided a buffer that was large enough for the message).

Thus the value of the *Reason* field on input to the exit cannot always be used to decide whether the message has been truncated.

The distinguishing characteristic of a truncated message is that the length provided to the exit in the *InBufferLength* parameter is *less than* the length implied by the format name contained in the *Format* field in the message descriptor. The exit should therefore check the value of *InBufferLength* before attempting to convert any of the data; the exit *should not* assume that the full amount of data implied by the format name has been provided.

If the exit has *not* been written to convert truncated messages, and *InBufferLength* is less than the value expected, the exit will return MQXDR_CONVERSION_FAILED in the *ExitResponse* field of the *DataConvExitParms* parameter, with the *CompCode* and *Reason* fields set to MQCC_WARNING and MQRC_FORMAT_ERROR.

If the exit *has* been written to convert truncated messages, the exit will convert as much of the data as possible (see next usage note), taking care not to attempt to examine or convert data beyond the end of *InBuffer*. If the conversion completes successfully, the exit will leave the *Reason* field in the *DataConvExitParms* parameter unchanged. This returns MQRC_TRUNCATED_MSG_ACCEPTED if the message was truncated by the receiver's queue manager, and MQRC_NONE if the message was truncated by the sender of the message.

It is also possible for a message to expand *during* conversion, to the point where it is bigger than *OutBuffer*. In this case the exit must decide whether to truncate the message; the *AppOptions* field in the *DataConvExitParms* parameter indicates whether the receiving application specified the MQGMO_ACCEPT_TRUNCATED_MSG option.

4. Generally, all the data in the message provided to the exit in *InBuffer* is converted, or that none of it is. An exception to this, however, occurs if the message is truncated, either before conversion or during conversion; in this case there can be an incomplete item at the end of the buffer (for example: 1 byte of a double-byte character, or 3 bytes of a 4-byte integer). In this situation, consider omitting the incomplete item and set the unused bytes in the *OutBuffer* to nulls. However, complete elements or characters within an array or string *should* be converted.
5. When an exit is needed for the first time, the queue manager attempts to load an object that has the same name as the format (apart from extensions). The object loaded must contain the exit that processes messages with that format name. Consider making the exit name, and the name of the object that contains the exit identical, although not all environments require this.
6. A new copy of the exit is loaded when an application attempts to retrieve the first message that uses that *Format* since the application connected to the queue manager. For CICS or IMS applications, this means when the CICS or IMS subsystem connected to the queue manager. A new copy can also be loaded at other times, if the queue manager has discarded a previously loaded copy. For this reason, an exit must not attempt to use static storage to communicate information from one invocation of the exit to the next - the exit can be unloaded between the two invocations.
7. If there is a user-supplied exit with the same name as one of the built-in formats supported by the queue manager, the user-supplied exit does not replace the built-in conversion routine. The only circumstances in which such an exit is invoked are:
   - If the built-in conversion routine cannot handle conversions to or from either the *CodedCharSetId* or *Encoding* involved, or
   - If the built-in conversion routine has failed to convert the data (for example, because there is a field or character which cannot be converted).
8. The scope of the exit is environment-dependent. *Format* names must be chosen to minimize the risk of clashes with other formats. Consider starting with characters that identify the application defining the format name.
9. The data-conversion exit runs in an environment like that of the program which issued the MQGET call; environment includes address space and user profile (where applicable). The program could be

a message channel agent sending messages to a destination queue manager that does not support message conversion. The exit cannot compromise the queue manager's integrity, since it does not run in the queue manager's environment.

10. The only MQI call which can be used by the exit is MQXCNVC; attempting to use other MQI calls fails with reason code MQRC_CALL_IN_PROGRESS, or other unpredictable errors.
11. No entry point called MQ_DATA_CONV_EXIT is provided by the queue manager. However, a **typedef** is provided for the name MQ_DATA_CONV_EXIT in the C programming language, and this can be used to declare the user-written exit, to ensure that the parameters are correct. The name of the exit must be the same as the format name (the name contained in the *Format* field in MQMD), although this is not required in all environments.

    The following example illustrates how the exit that processes the format MYFORMAT can be declared in the C programming language:

```
#include "cmqc.h"
#include "cmqxc.h"

MQ_DATA_CONV_EXIT MYFORMAT;

void MQENTRY MYFORMAT(
     PMQDXP   pDataConvExitParms, /* Data-conversion exit parameter
                                     block */
     PMQMD    pMsgDesc,           /* Message descriptor */
     MQLONG   InBufferLength,     /* Length in bytes of InBuffer */
     PMQVOID  pInBuffer,          /* Buffer containing the unconverted
                                     message */
     MQLONG   OutBufferLength,    /* Length in bytes of OutBuffer */
     PMQVOID  pOutBuffer)         /* Buffer containing the converted
                                     message */
{
  /* C language statements to convert message */
}
```

12. On z/OS, if an API-crossing exit is also in force, it is called after the data-conversion exit.

C invocation

```
exitname (&DataConvExitParms, &MsgDesc, InBufferLength,
         InBuffer, OutBufferLength, OutBuffer);
```

The parameters passed to the exit are declared as follows:

```
MQDXP   DataConvExitParms;  /* Data-conversion exit parameter block */
MQMD    MsgDesc;            /* Message descriptor */
MQLONG  InBufferLength;     /* Length in bytes of InBuffer */
MQBYTE  InBuffer[n];        /* Buffer containing the unconverted
                               message */
MQLONG  OutBufferLength;    /* Length in bytes of OutBuffer */
MQBYTE  OutBuffer[n];       /* Buffer containing the converted
                               message */
```

COBOL declaration (IBM i only)

```
     CALL 'exitname' USING DATACONVEXITPARMS, MSGDESC, INBUFFERLENGTH,
                           INBUFFER, OUTBUFFERLENGTH, OUTBUFFER.
```

The parameters passed to the exit are declared as follows:

```
**   Data-conversion exit parameter block
 01  DATACONVEXITPARMS.
     COPY CMQDXPV.
**   Message descriptor
 01  MSGDESC.
     COPY CMQMDV.
**   Length in bytes of INBUFFER
 01  INBUFFERLENGTH     PIC S9(9) BINARY.
**   Buffer containing the unconverted message
```

```
 01  INBUFFER           PIC X(n).
**   Length in bytes of OUTBUFFER
 01  OUTBUFFERLENGTH    PIC S9(9) BINARY.
**   Buffer containing the converted message
 01  OUTBUFFER          PIC X(n).
```

System/390 assembler declaration

```
        CALL EXITNAME,(DATACONVEXITPARMS,MSGDESC,INBUFFERLENGTH,      X
              INBUFFER,OUTBUFFERLENGTH,OUTBUFFER)
```

The parameters passed to the exit are declared as follows:

```
DATACONVEXITPARMS  CMQDXPA  ,      Data-conversion exit parameter block
MSGDESC            CMQMDA   ,      Message descriptor
INBUFFERLENGTH     DS       F      Length in bytes of INBUFFER
INBUFFER           DS       CL(n)  Buffer containing the unconverted
*                                  message
OUTBUFFERLENGTH    DS       F      Length in bytes of OUTBUFFER
OUTBUFFER          DS       CL(n)  Buffer containing the converted
*                                  message
```

Properties specified as MQRFH2 elements

Non-message descriptor properties can be specified as elements in MQRFH2 header folders. Overview of MQRFH2 elements being specified as properties.

This retains compatibility with the previous versions of the WebSphere MQ JMS and XMS clients. This section describes how to specify properties in MQRFH2 headers.

To use MQRFH2 elements as properties, specify the elements as described in Using WebSphere MQ classes for Java. This information supplements the information described in “MQRFH2 - Rules and formatting header 2” on page 1821.

Mapping property data types to MQRFH2 data types:

This topic provides information on message property types mapped to their corresponding MQRFH2 data types.

Table 227. Supported MQRFH2 data types

Message property type	MQRFH2 data type
MQBYTE[]	bin.hex
MQBOOL	boolean
MQINT8	i1
MQINT16	i2
MQINT32	i4
MQINT64	i8
MQFLOAT32	r4
MQFLOAT64	r8
MQCHAR[]	string

Any element without a data type is assumed to be of type "string".

An MQRFH2 data type of `int`, meaning an integer of unspecified size, is treated as if it were an `i8`.

A null value is indicated by the element attribute `xsi:nil='true'`. Do not use the attribute `xsi:nil='false'` for non-null values.

For example, the following property has a null value:

```
<NullProperty xsi:nil='true'></NullProperty>
```

A byte or character string property can have an empty value. This is represented by an MQRFH2 element with a zero length element value.

For example, the following property has an empty value:

```
<EmptyProperty></EmptyProperty>
```

Supported MQRFH2 folders:

Overview of the use of message descriptor fields as properties.

The folders <jms>, <mcd>, <mqext>, and <usr> are described in The MQRFH2 header and JMS. The <usr> folder is used to transport any JMS application-defined properties that are associated with a message. Groups are not allowed in the <usr> folder.

The MQRFH2 header and JMS supports the following additional folders:

- <mq>

  This folder is used and reserved for MQ-defined properties that are used by IBM WebSphere MQ.
- <mq_usr>

  This folder can be used to transport any application-defined properties that are not exposed as JMS user-defined properties, as the properties might not meet the requirements of a JMS property. This folder can contain groups that the <usr> folder cannot.
- Any folder marked with the content='properties' attribute.

  Such a folder is equivalent to the <mq_usr> folder in content.
- <mqps>

  This folder is used for IBM WebSphere MQ publish/subscribe properties.

IBM WebSphere MQ also supports the following folders that are already in use by WAS/SIB:

- <sib>

  This folder is used and reserved for WAS/SIB system message properties that are not exposed as JMS properties, or are mapped to JMS_IBM_* properties, but are exposed to WAS/SIB applications; these include forward and reverse routing paths properties.

  At least some cannot be exposed as JMS properties, because they are byte arrays. If your application adds properties to this folder, the value is either ignored or removed.
- <sib_usr>

  This folder is used and reserved for WAS/SIB user message properties that cannot be exposed as JMS user properties because they are not of supported types; they are exposed to WAS/SIB applications.

  These are user properties, that you can get or set through the SIMessage interface, but the content of the byte array is mapped to the required property value.

  If your IBM WebSphere MQ application writes an arbitrary bin.hex element to the folder, the application probably receives an IOException, as it is not of the format expected to restore. If you add anything other than a bin.hex element you receive a ClassCastException.

  Do not attempt to make properties available to WAS/SIB by using this folder; instead user the <usr> folder for that purpose.
- <sib_context>

  This folder is used for WAS/SIB system message properties that are not exposed to WAS/SIB user applications or as JMS properties. These include security and transactional properties that are used for web services and similar.

  Your application must not add properties to this folder.

- <mqema>
  This folder was used by WAS/SIB instead of the <mqext> folder.

MQRFH2 folder names are case-sensitive.

The following folders are reserved, in any mixture of lowercase or uppercase characters:
- Any folder prefixed by mq or wmq; reserved for use by IBM WebSphere MQ.
- Any folder prefixed by sib; reserved for use by WAS/SIB.
- <Root> and <Body> folders; reserved but not used.

The following folders are not recognized as containing message properties:
- <psc>
  Used by WebSphere Message Broker to convey publish/subscribe command messages to the broker.
- <pscr>
  Used by WebSphere Message Broker to contain information from the broker, in response to publish/subscribe command messages.
- Any folder not defined by IBM, that is not marked with the content='properties' attribute.

Do not specify content='properties' on the <psc> or <pscr> folders. If you do so, these folders are treated as properties and WebSphere Message Broker is likely to stop functioning as expected.

If your application is building messages with properties, in MQRFH2 headers to be recognized as an MQRFH2 header containing properties, the header must be in the list of headers that can be chained at the head of the message.

The MQRFH2 can be preceded by any number of MQH standard headers, or an MQCIH, an MQDLH, an MQIIH, an MQTM, an MQTMC2, or an MQXQH. A string or an MQCFH ends parsing because they cannot be chained.

It is possible for a message to contain multiple MQRFH2 headers all carrying message properties. Folders with the same name can coexist in different headers unless otherwise restricted, for example by WAS/SIB. The folders are treated as one logical folder, if they are all in significant headers.

While folders from the significant headers cannot be merged with those folders in nonsignificant headers, folders with the same name within the significant headers can be merged, removing any conflicting properties. Your applications must not depend on the layout of properties within their message.

MQRFH2 groups are parsed for properties in user-defined folders, that is, not the <wmq>, <jms>, <mcd>, <usr>, <mqext>, <sib>, <sib_usr>, <sib_context>, and <mqema> folders.

Groups in the IBM-defined properties folders, except for the <wmq> and <mq> folders, are parsed for properties.

An MQRFH2 folder cannot contain mixed content; a folder or group can contain either groups or properties, or a value, but not both.

A segment of a message, either the first or a subsequent segment, cannot contain IBM WebSphere MQ-defined properties other than those properties in the message descriptor. Therefore putting a message containing such properties with either MQMF_SEGMENT or MQMF_SEGMENTATION_ALLOWED set causes the put to fail with MQRC_SEGMENTATION_NOT_ALLOWED.

However, message groups can contain IBM WebSphere MQ-defined properties.

Generation of MQRFH2 headers:

If WebSphere MQ converts message properties to their MQRFH2 representation, it must add the MQRFH2 to the message. It adds the MQRFH2 either as a separate header, or merges it with an existing header.

Generation of new MQRFH2 headers by WebSphere MQ might disrupt existing headers in a message. Applications that parse a message buffer for headers must be aware that the number and position of headers in a buffer might change in some circumstances. WebSphere MQ attempts to minimize the impact of adding properties to a message by merging message properties into an existing MQRFH2 header, where it can. It also attempts to minimize the impact by inserting a generated MQRFH2 into a fixed position relative to other headers in the message buffer.

A generated MQRFH2 header is placed following the MQMD, and any number of MQXQH, MQRFH, and MQDLH headers, whatever order they are in. The generated MQRFH2 header is placed immediately before the first header that is not an MQMD, MQXQH, MQDLH, or MQRFH header.

Rules for merging generated MQRFH2

The following rules apply to merging a generated MQRFH2 with an existing MQRFH2. The generated MQRFH2 header is merged with an existing MQRFH2 header, if:

1. The existing MQRFH2 is in the same position WebSphere MQ would place a generated MQRFH2, or earlier in the header chain.
2. The CCSID of the generated properties is the same as the NameValueCCSID of the existing MQRFH2.

Otherwise, the generated header is placed separately in the buffer, in the position described before.

Rules for merging folders in an existing MQRFH2

If message properties are merged into an existing MQRFH2, then the existing MQRFH2 is scanned for folders that match the message properties, and merges them. If a matching folder does not exist a new folder is added to the end of the existing folders. If a matching folder does exist, the folder is searched. Any matching properties are overwritten. Any new ones are added at the end of the folder.

MQRFH2 folder restrictions:

Overview of folder restrictions in MQRFH2 headers

The MQRFH2 restrictions apply to the following folders:

- Element names in the <usr> folder must not begin with the prefix JMS; such property names are reserved for use by JMS and are not valid for user-defined properties.

  Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the WebSphere MQ message property APIs.
- Element names in the <usr> folder must not be, in any mixture of lower or uppercase, NULL, TRUE, FALSE, NOT, AND, OR, BETWEEN, LIKE, IN, IS and ESCAPE. These names match SQL keywords and make parsing selectors harder, because <usr> is the default folder used when no folder is specified for a particular property in a selector.

  Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the WebSphere MQ message property APIs.
- Element names in any folder considered to contain message properties must not contain a period (.) (Unicode character U+002E), because this is used in property names to indicate the hierarchy.

  Such an element name does not cause parsing of the MQRFH2 to fail, but is not accessible to the WebSphere MQ message property APIs.

In general, MQRFH2 headers that contain valid XML-style data can be parsed by WebSphere MQ without failure, although certain elements of the MQRFH2 are not accessible through the WebSphere MQ message property APIs.

MQRFH2 element name conflicts:

Overview of conflicts within MQRFH2 element names.

Only one value can be attached to a message property. If an attempt to access a property leads to a conflict of values, one is chosen in preference over another.

The WebSphere MQ syntax for accessing MQRFH2 elements allows unique identification of an element, if a folder contains no elements with the same name. If a folder contains more than one element with the same name, the value of the property used is the one closest to the head of the message.

This applies if two or more folders of the same name are contained in different significant MQRFH2 headers within the same message.

A conflict can result when the MQGET call is processed after a non-message descriptor property has been set twice: both through an MQSETMP call and directly in the raw MQRFH2 header.

If this happens, the property associated with the message by an API call takes preference over one in the message data, that is, the one in the raw MQRFH2 header. If a conflict occurs, it is considered to come logically before the message data.

Mapping from property names to MQRFH2 folder and element names:

Overview of the differences between property names and element names in the MQRFH2 header.

When using any of the defined APIs that ultimately generate MQRFH2 headers, in order to specify message properties (for example, MQ JMS), the property name is not necessarily the element name in the MQRFH2 folder.

Therefore, a mapping occurs from the property name to the MQRFH2 element, and in the reverse way, taking into account both the folder name that contains the element, and the element name. Some examples from IBM WebSphere MQ classes for JMS are already documented in Using Java.

Property name	MQRFH2 folder name	MQRFH2 element name
`JMSDestination`	`jms`	`Dst`
`JMSType`	`mcd`	`Type`, `Set`, `Fmt`
`xxx` (user defined, where `xxx` does not begin with JMS)	`usr`	`xxx`

Therefore, when a JMS application accesses the `JMSDestination` property this maps to the `Dst` element in the `<jms>` folder.

When specifying properties as MQRFH2 elements, IBM WebSphere MQ defines its elements as follows:

Property name	MQRFH2 folder name	MQRFH2 group name	MQRFH2 element name
<Property>	<usr>	n/a	<Property>
<folder>.<Property>	<folder>	n/a	<Property>
<folder>.<group>.<Property>	<folder>	<group>	<Property>

For example, when a IBM WebSphere MQ application attempts to access the Property1 property, this maps to the Property1 element in the <usr> folder. The wmq.Property2 property maps to the Property2 property in the <wmq> folder.

If the property name contains more than one . character, the MQRFH2 element name used is the one following the final . character, and MQRFH2 groups are used to form a hierarchy; nested MQRFH2 groups are permitted.

The JMS header and provider-specific properties that are contained in an MQRFH2 in the <mcd>, <jms>, and <mqext> folders are accessed by a IBM WebSphere MQ application using the short names defined in Using WebSphere MQ classes for Java.

JMS user-defined properties are accessed from the <usr> folder. A IBM WebSphere MQ application can use the <usr> folder for its application properties if it is acceptable for the property to appear to JMS applications as one of its user-defined properties.

If it is not acceptable, choose another folder; the <wmq_usr> folder is provided as a standard location for such non-JMS properties.

Your applications can specify and use any MQRFH2 folder with a well-defined use, not documented in "Properties specified as MQRFH2 elements" on page 2233 if you note the following:

1. The folder might already be in use, or might be used in the future, by another application providing undefined access to properties contained inside it; see Property names for the suggested naming convention for property names.
2. The properties are not accessible to previous versions of the IBM WebSphere MQ classes for JMS or XMS client that can only access the <usr> folder for user-defined properties
3. The folder must be marked with the attribute content with the value set to properties, for example, content='properties'.

   "MQSETMP - Set message property" on page 2078 automatically adds this attribute as required. This attribute must not be added to any of the IBM-defined folders, for example, <jms> and <usr>. Doing so, causes the message to be rejected by the IBM WebSphere MQ classes for JMS client before Version 7.0. with a MessageFormatException.

Because the <usr> folder is the default location for properties of the <Property> syntax, a IBM WebSphere MQ application and a JMS application to access the same user-defined property value using the same name.

Reserved folder names

There are several reserved folder names. You cannot use such names as your folder prefixes; for example, Root.Property1 does not access a valid property because Root is reserved. The following list contains reserved folder names:

- Root
- Body
- Properties
- Environment
- LocalEnvironment

- DestinationList
- ExceptionList
- InputBody
- InputRoot
- InputProperties
- InputLocalEnvironment
- InputDestinationList
- InputExceptionList
- OutputRoot
- OutputLocalEnvironment
- OutputDestinationList
- OutputExceptionList

Mapping property descriptor fields into MQRFH2 headers:

When a property is translated into an MQRFH2 element the following element attributes are used to specify the significant fields of the property descriptor: This describes how MQPD fields are translated to MQRFH2 element attributes.

Support

The Support property descriptor field is split into three element attributes
- The **sr** element attribute specifies values in the MQPD_REJECT_UNSUP_MASK bit mask.
- The **sa** element attribute specifies values in the MQPD_ACCEPT_UNSUP_MASK bit mask.
- The **sx** element attribute specifies values in the MQPD_ACCEPT_UNSUP_IF_XMIT_MASK bit mask.

These element attributes are only valid in the <mq> folder and are ignored if set on elements in the other folders containing properties.

Support value	MQRFH2 element attribute	MQRFH2 attribute value
MQPD_SUPPORT_OPTIONAL	sa	optional This is the default value.
MQPD_SUPPORT_REQUIRED	sr	required
MQPD_SUPPORT_REQUIRED_IF_LOCAL	sx	local

Context

Use the **context** element attribute to indicate the message context to which a property belongs. Use one value only. This element attribute is valid on a property in any folder containing properties.

Context value	MQRFH2 attribute value
MQPD_NO_CONTEXT	none This is the default value.
MQPD_USER _CONTEXT	user

CopyOptions

Use the **copy** element attribute to indicate messages into which a property should be copied. More than one value is acceptable; separate multiple values with a comma. For example **copy='reply'** and **copy='publish,report'** are both valid. This element attribute is valid on a property in any folder containing properties.

Note: In the attribute definition, single quotation marks or double quotation marks are valid use, for example **copy='reply'** or **copy="report"**

CopyOption value	MQRFH2 attribute value
MQPD_COPY_FORWARD	forward
MQPD_COPY_REPLY	reply
MQPD_COPY_REPORT	report
MQPD_COPY_PUBLISH	publish
MQPD_COPY_ALL	all Do not specify this with any other value. When used with another value, this takes precedence over any value except **none**.
MQPD_COPY_DEFAULT	default This is the default value. It is equivalent to specifying the three values MQCOPY_FORWARD, MQCOPY_REPORT and MQCOPY_PUBLISH. Do not specify this with any other value.
MQPD_COPY_NONE	none Do not specify this with any other value. When used with another value, this takes precedence.

Restrictions to the <mq> MQRFH2 folder

When a message is put on to a queue, it is searched for an <mq> folder so that the message can be processed according to its MQ-defined properties. To allow the efficient parsing of MQ-defined properties, the following restrictions apply to the folder:

- Only properties in the first significant <mq> folder in the message are acted upon by MQ; properties in any other <mq> folder in the message are ignored.
- If the folder is in UTF-8, only single-byte UTF-8 characters are allowed in the folder. A multi-byte character in the folder, can cause parsing to fail, and the message to be rejected.
- Do not include MQRFH2 groups in the <mq> folder. The presence of Unicode character U+003C in a property value will cause the message to be rejected.
- Do not use escape strings in the folder. An escape string is treated as the actual value of the element.
- Only Unicode character U+0020 is treated as white space within the folder. All other characters are treated as significant and can cause parsing of the folder to fail, and the message to be rejected.

If parsing of the <mq> folder fails, or if the folder does not observe these restrictions, the message is rejected with CompCode **MQCC_FAILED** and Reason **MQRC_RFH_RESTRICTED_FORMAT_ERR**.

MQRFH2 headers that are not valid:

At the time an MQPUT, MQPUT1, or MQGET call processes, a partial parsing of any MQRFH2 headers in the message can occur to check what folders are included, and to determine if the folders contain properties. Overview of MQRFH2 headers that are not valid.

If the partial parsing of the message cannot complete successfully because the structure is not valid, for example, the `StrucLength` field is too small, then:

- The MQPUT or MQPUT1 call fails with reason code MQRC_RFH_ERROR, if it can be determined that the application includes some WebSphere MQ Version 7 option, so that existing applications do not fail.
- The MQGET call returns successfully, and the MQRFH2 containing the error is returned in the buffer you provided.

If the partial parsing fails because it cannot be detected whether a particular folder contains properties or not, for example, the folder begins `<<jms`, so parsing fails before the folder name is determined, then:

- The MQPUT or MQPUT1 call fails with reason code MQRC_RFH_FORMAT_ERROR, if it can be determined that the application includes some WebSphere MQ Version 7 option, so that existing applications do not fail.
- The MQGET call returns successfully, and the MQRFH2 containing the error is returned in the buffer you provided.
- While internally within the queue manager, the message is not rejected due to the badly formatted folder, but the folder is always treated as if no properties were contained inside it.

A message can flow through the queue manager network with a folder containing such a syntax error, but never being parsed and detected, while one or more folders in the message are:

- Valid
- Successfully parsed
- Used in the processing of the message

Therefore, detection is not guaranteed.

If one of your applications uses "MQSETMP - Set message property" on page 2078, or MQINQMP to access a property, and in so doing this causes an MQRFH2 folder to be fully parsed, detecting an error such that parsing cannot complete, this is indicated by an appropriate return code to the API call. No properties in the folder are made available to the application.

If an attempt is made to fully parse an MQRFH2 folder and the parser finds unrecognized element attributes, or an unrecognized data type, parsing continues and complete successfully with no warnings being issued; this does not constitute a parsing error.

Code page conversion

This section describes codeset names and CCSIDs, national language, z/OS conversion, IBM i conversion, and Unicode conversion support.

Each national language section lists the following information:

- The native CCSIDs supported
- The code page conversions that are **not** supported

The following terms are used in the information:

-8 Indicates for HP-UX that the CCSID is for the HP-UX defined codeset *roman8*

AIX Indicates WebSphere MQ for AIX

HP-UX
Indicates WebSphere MQ for HP-UX

Linux Indicates WebSphere MQ for Linux for Intel and WebSphere MQ for Linux for zSeries

HP Integrity NonStop Server
Indicates WebSphere MQ for HP Integrity NonStop Server

OS/400
Indicates WebSphere MQ for IBM i

Solaris
Indicates WebSphere MQ for Solaris

Windows
Indicates WebSphere MQ for Windows

z/OS Indicates WebSphere MQ for z/OS

The default for data conversion is for the conversion to be performed at the target (receiving) system.

If the source product supports the conversion a channel can be set up and data exchanged by setting the channel attribute **DataConversion** to YES at the source.

Note:

1. Conversion for WebSphere MQ MQI client information takes place in the server, so the server must support conversion from the client CCSID to the server CCSID.
2. The conversion might include support added by CSD/PTF to the latest version of WebSphere MQ. Check the content of the latest service level to see if you need to install a CSD/PTF to enable this conversion.

See Table 228 for a cross reference between some of the CCSID numbers and some industry codeset names.

Codeset names and CCSIDs:

WebSphere MQ for z/OS provides more conversion than is listed in the language specific tables.

Table 228. Codeset names and CCSIDs

Codeset names	CCSIDs
ISO 8859-1	819
ISO 8859-2	912
ISO 8859-3	913
ISO 8859-5	915
ISO 8859-6	1089
ISO 8859-7	813
ISO 8859-8	916
ISO 8859-9	920
ISO 8859-13	921
ISO 8859-15 (euro)	923
big5	950
eucJP	954 5050 33722
eucKR	970
eucTW	964

Table 228. Codeset names and CCSIDs (continued)

Codeset names	CCSIDs
eucCN	1383
PCK	943
GBK	1386
koi8-r	878

National languages:

This information contains languages supported by WebSphere MQ.

The languages supported by WebSphere MQ are:

- US English - see topic "US English" on page 2244
- German - see topic "German" on page 2244
- Danish and Norwegian - see topic "Danish and Norwegian" on page 2245
- Finnish and Swedish - see topic "Finnish and Swedish" on page 2245
- Italian - see topic "Italian" on page 2246
- Spanish - see topic "Spanish" on page 2247
- UK English / Gaelic - see topic "UK English /Gaelic" on page 2247
- French - see topic "French" on page 2248
- Multilingual - see topic "Multilingual" on page 2248
- Portuguese - see topic "Portuguese" on page 2249
- Icelandic - see topic "Icelandic" on page 2250
- Eastern European languages - see topic "Eastern European languages" on page 2250
- Cyrillic - see topic "Cyrillic" on page 2252
- Estonian - see topic "Estonian" on page 2252
- Latvian and Lithuanian - see topic "Latvian and Lithuanian" on page 2253
- Ukranian - see topic "Ukrainian" on page 2254
- Greek - see topic "Greek" on page 2255
- Turkish - see topic "Turkish" on page 2255
- Hebrew - see topic "Hebrew" on page 2256
- Farsi - see topic "Farsi" on page 2258
- Urdu - see topic "Urdu" on page 2258
- Thai - see topic "Thai" on page 2258
- Lao - see topic "Lao" on page 2259
- Vietnamese - see topic "Vietnamese" on page 2259
- Japanese Latin SBCS - see topic "Japanese Latin SBCS" on page 2259
- Japanese Katakana SBCS - see topic "Japanese Katakana SBCS" on page 2261
- Japanese Kanji/ Latin Mixed - see topic "Japanese Kanji/ Latin Mixed" on page 2262
- Japanese Kanji/ Katakana Mixed - see topic "Japanese Kanji/ Katakana Mixed" on page 2263
- Korean - see topic "Korean" on page 2265
- Simplified Chinese - see topic "Simplified Chinese" on page 2265
- Traditional Chinese - see topic "Traditional Chinese" on page 2266

US English:

Details of CCSIDs and CCSID conversion for US English.

The following table shows the native CCSIDs for US English on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	37, 924, 1140
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 1252, 5348, 858
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

37 Does not convert to code pages 923, 858

924 Does not convert to code pages 437, 858, 1051, 1140, 1252, 1275, 5348

1140 Does not convert to code pages 924, 1051, 1275

German:

Details of CCSIDs and CCSID conversion for German.

The following table shows the native CCSIDs for German on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	273, 924, 1141
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

273 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 273, 437, 858, 1051, 1141, 1252, 1275, 5348

1141 Does not convert to code pages 924, 1051, 1275

Danish and Norwegian:

Details of CCSIDs and CCSID conversion for Danish and Norwegian.

The following table shows the native CCSIDs for Danish and Norwegian on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	277, 924, 1142
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	850, 858, 865, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

277 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 277, 858, 865, 1051, 1142, 1252, 1275, 5348

1142 Does not convert to code pages 924, 865, 1051, 1275

AIX

Code page:

819 Does not convert to code page 865

HP-UX

Code page:

1051 Does not convert to code page 865

Windows

Code page:

865 Does not convert to code pages 1051, 1275

Finnish and Swedish:

Details of CCSIDs and CCSID conversion for Finnish and Swedish.

The following table shows the native CCSIDs for Finnish and Swedish on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	278, 924, 1143
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 865, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

278 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 278, 437, 858, 865, 1051, 1143, 1252, 1275, 5348

1143 Does not convert to code pages 865, 924, 1051, 1275

AIX

Code page:

819 Does not convert to code page 865

850 Does not convert to code page 865

HP-UX

Code page:

1051 Does not convert to code page 865

Windows

Code page:

865 Does not convert to code pages 1051, 1275

Italian:

Details of CCSIDs and CCSID conversion for Italian.

The following table shows the native CCSIDs for Italian on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	280, 924, 1144
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

280 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 280, 437, 858, 1051, 1144, 1252, 1275, 5348

1144 Does not convert to code pages 924, 1051, 1275

Spanish:

Details of CCSIDs and CCSID conversion for Spanish.

The following table shows the native CCSIDs for Spanish on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	284, 924, 1145
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

284 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 284, 437, 858, 1051, 1145, 1252, 1275, 5348

1145 Does not convert to code pages 924, 1051, 1275

UK English /Gaelic:

Details of CCSIDs and CCSID conversion for UK English/Gaelic.

The following table shows the native CCSIDs for UK English / Gaelic on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	285, 924, 1146
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

285 Does not convert to code pages 858, 923, 924, 1275

924 Does not convert to code pages 285, 437, 858, 1051, 1146, 1252, 1275, 5348

1146 Does not convert to code pages 924, 1051, 1275

French:

Details of CCSIDs and CCSID conversion for French.

The following table shows the native CCSIDs for French on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	297, 924, 1147
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

297 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 297, 437, 858, 1051, 1147, 1252, 1275, 5348

1147 Does not convert to code pages 924, 1051, 1275

Multilingual:

Details of CCSIDs and CCSID conversion for Multilingual.

The following table shows the native CCSIDs for multilingual conversion on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	500, 924, 1148
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	437, 850, 858, 1252, 5348
HP Integrity NonStop Server, SINIX, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

500 Does not convert to code pages 858, 923

924 Does not convert to code pages 437, 858, 1051, 1148, 1252, 1275, 5348

1148 Does not convert to code pages 924, 1051, 1275

Portuguese:

Details of CCSIDs and CCSID conversion for Portuguese.

The following table shows the native CCSIDs for Portuguese on supported platforms:

Platform	Native CCSIDs
IBM i	37, 500, 924, 1140
z/OS	500, 924, 1140
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	850, 858, 860, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

37 Does not convert to code pages 858, 923, 1275

500 Does not convert to code pages 858, 923, 1275

924 Does not convert to code pages 858, 860, 1051, 1140, 1252, 1275, 5348

1140 Does not convert to code pages 860, 924, 1051, 1275

HP-UX

Code page:

1051 Does not convert to code page 860

Windows

Code page:

860 Does not convert to code pages 1051, 1275

Icelandic:

Details of CCSIDs and CCSID conversion for Icelandic.

The following table shows the native CCSIDs for Icelandic on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	871, 924, 1149
AIX	819, 923, 5348
HP-UX	819, 923, 1051
Windows	850, 858, 861, 1252, 5348
HP Integrity NonStop Server, Solaris, Linux	819, 923
Apple client	1275

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

871 Does not convert to code pages 858, 923, 924, 1275, 5348

924 Does not convert to code pages 858, 861, 871, 1051, 1149, 1252, 1275, 5348

1149 Does not convert to code pages 924, 1051, 1275

HP-UX

Code page:

1051 Does not convert to code page 861

Windows

Code page:

861 Does not convert to code pages 1051, 1275

Eastern European languages:

Details of CCSIDs and CCSID conversion for Eastern European Languages. The typical languages using these CCSIDs include Albanian, Croatian, Czech, Hungarian, Polish, Romanian, Serbian, Slovak, and Slovenian.

The following table shows the native CCSIDs for Eastern European languages on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	870, 1153
Windows	852, 1250, 5346, 9044
AIX, HP-UX, HP Integrity NonStop Server, Solaris, Linux	912
Eastern European Apple client	1282
Romanian Apple client	1285
Croatian Apple client	1284

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

z/OS

Code page:

870 Does not convert to code pages 1284, 1285

1153 Does not convert to code pages 1250, 1284, 1285

IBM i

Code page:

870 Does not convert to code pages 1284, 1285, 5346, 9044

1153 Does not convert to code pages 1282, 1284, 1285, 5346, 9044

HP-UX, Solaris, Linux

Code page:

912 Does not convert to code pages 1284, 1285

HP Integrity NonStop Server

Code page:

912 Does not convert to code pages 1153, 1284, 1285, 9044

Windows

Code page:

852 Does not convert to code pages 1284, 1285

1250 Does not convert to code pages 1284, 1285

9044 Does not convert to code pages 912, 1282, 1284, 1285

Cyrillic:

Details of CCSIDs and CCSID conversion for Cyrillic. The typical languages using these CCSIDs include Belarussion, Bulgarian, Macedonian, Russian, and Serbian.

The following table shows the native CCSIDs for Cyrillic on supported platforms:

Platform	Native CCSIDs
z/OS	1025
IBM i	880, 1025
Windows	855, 866, 1131, 1251, 5347
Solaris	878, 915
AIX, HP-UX, Linux, HP Integrity NonStop Server	915
Apple client	1283

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

880 Does not convert to code pages 855, 866, 878, 1131, 5347

1025 Does not convert to code pages 878, 5347

Windows

Code page:

855 Does not convert to code page 1131

866 Does not convert to code page 1131

1131 Does not convert to code pages 855, 866, 880, 1283

Estonian:

Details of CCSIDs and CCSID conversion for Estonian.

The following table shows the native CCSIDs for Estonian on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1122, 1157
Windows	902, 922, 1257, 5353, 9449
AIX, HP-UX, Solaris, Linux	902, 922
HP Integrity NonStop Server	922

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

1122 Does not convert to code pages 902, 1157, 9449

1157 Does not convert to code pages 922, 1122, 1257, 9449

IBM i

Code page:

1122 Does not convert to code pages 902, 5353, 9449

1157 Does not convert to code pages 922, 5353, 9449

HP-UX, Solaris, Linux

Code page:

902 Does not convert to code pages 922, 1122, 9449

922 Does not convert to code pages 902, 1157, 9449

Windows

Code page:

5353 Does not convert to code page 9449

9449 Does not convert to code pages 902, 922, 1122, 1157, 1257, 5353

902 Does not convert to code pages 922, 1122, 9449

HP Integrity NonStop Server

Code page:

922 Does not convert to code pages 902, 1157, 9449

Latvian and Lithuanian:

Details of CCSIDs and CCSID conversion for Latvian and Lithuanian.

The following table shows the native CCSIDs for Latvian and Lithuanian on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1112, 1156
Windows	901, 921, 1257, 5353, 9449
AIX, HP-UX, Solaris, Linux	901, 921
HP Integrity NonStop Server	921

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

1112 Does not convert to code pages 901, 1156, 9449

1156 Does not convert to code pages 901, 1156, 9449

IBM i

Code page:

1112 Does not convert to code page 5353

1153 Does not convert to code pages 921, 5353, 9449

HP-UX, Solaris, Linux

Code page:

902 Does not convert to code pages 921, 1112, 1257, 9449

921 Does not convert to code pages 901, 1156, 9449

Windows

Code page:

901 Does not convert to code pages 921, 1112, 1257, 9449

5355 Does not convert to code page 9449

9449 Does not convert to code pages 901, 921, 1112, 1156, 1257

HP Integrity NonStop Server

Code page:

921 Does not convert to code pages 901, 1156, 9449

Ukrainian:

Details of CCSIDs and CCSID conversion for Ukrainian.

The following table shows the native CCSIDs for Ukranian on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1123
Windows	1124, 1125, 1251, 5347
AIX, HP-UX, HP Integrity NonStop Server, Solaris, Linux	1124

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

IBM i

Code page:

1123 Does not convert to code page 5347

HP-UX

Code page:

1124 Does not convert to code page 5347

Windows

Code page:

1125 Does not convert to code page 1123

Greek:

Details of CCSIDs and CCSID conversion for Greek.

The following table shows the native CCSIDs for Greek on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	875
HP-UX	813 (see note)
Windows	869, 1253, 5349
AIX, NCR, HP Integrity NonStop Server, Solaris, Linux	813
Apple client	1280
DOS client	737
Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.	

All non-client platforms support conversion between their native CCSIDs, the native CCSIDs of the other platforms with the following exceptions.

IBM i

Code page:

875 Does not convert to code page 5349

Windows

Code page:

1253 Does not convert to code page 737

5349 Does not convert to code page 737

Turkish:

Details of CCSIDs and CCSID conversion for Turkish.

The following table shows the native CCSIDs for Turkish on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1026
HP-UX	920 (see note)
Windows	857, 1254, 5350
AIX, HP Integrity NonStop Server, Solaris, Linux	920
Apple client	1281
Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary turkish8 codeset has no registered CCSID and is not supported.	

All non-client platforms support conversion between their native CCSIDs and the native CCSIDs of the other platforms, with the following exceptions.

IBM i

Code page:

1026 Does not convert to code page 5350

Hebrew:

Details of CCSIDs and CCSID conversion for Hebrew.

The following table shows the native CCSIDs for Hebrew on supported platforms:

Platform	Native CCSIDs
z/OS	424, 803, 4899, 12712
IBM i	424
AIX	916, 9048
HP-UX	916 (see note)
Windows	1255, 5351
HP Integrity NonStop Server, Solaris, Linux	916
Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary greek8 codeset has no registered CCSID and is not supported.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

424 Does not convert to code pages 867, 4899, 9048, 12712

803 Does not convert to code pages 867, 4899, 5351, 9048, 12712

4899 Does not convert to code pages 424, 803, 856, 862, 916, 1255

12712 Does not convert to code pages 424, 803, 856, 916, 1255

IBM i

Code page:

424 Does not convert to code pages 803, 867, 4899, 5351, 9048, 12712

Code page 424 also converts to and from CCSID 4952, which is a variant of 856.

AIX

Code page:

916 Does not convert to code pages 867, 4899, 9048, 12712

9048 Does not convert to code pages 424, 803, 856, 862, 916, 1255

Windows

Code page:

1255 Does not convert to code pages 867, 4899, 9048, 12712

5351 Does not convert to code page 803

Arabic:

Details of CCSIDs and CCSID conversion for Arabic

The following table shows the native CCSIDs for Arabic on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	420
AIX	1046, 1089
HP-UX	1089 (see note)
Windows	720, 864, 1256, 5352
HP Integrity NonStop Server, Solaris, Linux	1089
Note: Only the ISO codeset is supported on HP-UX. The HP-UX proprietary arabic8 codeset has no registered CCSID and is not supported.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

IBM i

Code page:

420 Does not convert to code page 5352

HP-UX, Solaris, Linux, HP Integrity NonStop Server, Tru64

Code page:

1089 Does not convert to code page 720

Windows

Code page:

720 Does not convert to code pages 1089, 5352

5352 Does not convert to code page 720

Farsi:

Details of CCSIDs and CCSID conversion for Farsi.

The following table shows the native CCSIDs for Farsi on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1097
AIX, HP-UX, HP Integrity NonStop Server, Solaris, Linux Windows	1098 (see note)
Note: The native CCSID for these platforms has not been standardized and might change.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Urdu:

Details of CCSIDs and CCSID conversion for Urdu.

The following table shows the native CCSIDs for Urdu on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	918
Windows	868
AIX, HP-UX, HP Integrity NonStop Server, Solaris, Linux	1006

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

IBM i

Code page:

918 Does not convert to code page 1006

Thai:

Details of CCSIDs and CCSID conversion for Thai.

The following table shows the native CCSIDs for Thai on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	838
AIX, HP-UX, HP Integrity NonStop Server, Solaris, Linux Windows	874 (see note)
Note: The native CCSID for these platforms has not been standardized and might change.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Lao:

Details of CCSIDs and CCSID conversion for Lao.

The following table shows the native CCSIDs for Lao on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1132
AIX, HP-UX, HP Integrity NonStop Server, Solaris, Linux Windows	1133

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms.

Vietnamese:

Details of CCSIDs and CCSID conversion for Vietnamese.

The following table shows the native CCSIDs for Vietnamese on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1130
Windows	1258, 5354
AIX, HP-UX, HP Integrity NonStop Server, Solaris, Linux	1129

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

IBM i

Code page:

1130 Does not convert to code pages 1129, 5354

Japanese Latin SBCS:

Details of CCSIDs and CCSID conversion for Japanese Latin SBCS.

The following table shows the native CCSIDs for Japanese Latin SBCS on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1027
AIX	932, 5050, 33722 (see Note 1)
Windows	932, 943 (see Notes 2 and 3)
Linux, HP Integrity NonStop Server, Solaris	943, 5050
HP-UX	Not known

Platform	Native CCSIDs
Note: 1. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722. 2. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID. On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943. 3. WebSphere MQ does not support code pages based on the JIS X 0213 (JIS2004) standard.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

1027 Does not convert to code pages 932, 942, 943, 954, 5050, 33722

IBM i

Code page:

1027 Does not convert to code page 932

AIX

Code page:

932 Does not convert to code page 1027

5050 Does not convert to code page 1027

33722 Does not convert to code page 1027

Linux

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

Solaris

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

HP Integrity NonStop Server

Code page:

943 Does not convert to code page 1027

5050 Does not convert to code page 1027

Japanese Katakana SBCS:

Details of CCSIDs and CCSID conversion for Japanese Katakana SBCS.

The following table shows the native CCSIDs for Japanese Katakana SBCS on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	290
HP-UX	897
AIX	932, 5050, 33722 (see Note 1)
Windows	932, 943 (see Notes 2 and 3)
Linux, HP Integrity NonStop Server, Solaris	943, 5050

Note:

1. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
2. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.

   On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.
3. WebSphere MQ does not support code pages based on the JIS X 0213 (JIS2004) standard.
4. In addition to the above conversions, the WebSphere MQ products on AIX, HP-UX, Solaris, Linux and Tru64 support conversion from CCSID 897 to CCSIDs 37, 273, 277, 278, 280, 284, 285, 290, 297, 437, 500, 819, 850, 1027, and 1252.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

290 Does not convert to code pages 932, 943, 954, 5050, 33722

IBM i

Code page:

290 Does not convert to code page 932

AIX

Code page:

932 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

33722 Does not convert to code pages 290, 897

HP-UX

Code page:

897 Does not convert to code pages 932, 943, 954, 5050, 33722

Linux

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

Solaris

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

HP Integrity NonStop Server

Code page:

943 Does not convert to code pages 290, 897

5050 Does not convert to code pages 290, 897

Japanese Kanji/ Latin Mixed:

Details of CCSIDs and CCSID conversion for Japanese Kanji/Latin Mixed.

The following table shows the native CCSIDs for Japanese Kanji/ Latin Mixed on supported platforms:

Platform	Native CCSIDs
IBM i, z/OS	1399, 5035 (see Note 1)
AIX	932, 5050, 33722 (see Note 2)
HP-UX	932, 954, 5039 (see Note 3)
Windows	932, 943 (see Notes 4 and 5)
Linux, HP Integrity NonStop Server, Solaris	943, 5050

Note:

1. 5035 is a CCSID related to code page 939
2. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
3. Code sets `japan15` and SJIS on HP-UX are represented by CCSID 932. These have a few DBCS characters having different representations in SJIS so 932 may be converted incorrectly if the conversion is not performed on an HP-UX system. WebSphere MQ for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to file `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039.
4. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.

   On WebSphere MQ for Windows CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 943.
5. WebSphere MQ does not support code pages based on the JIS X 0213 (JIS2004) standard.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

1399 Does not convert to code pages 954, 5035, 5050, 33722

5035 Does not convert to code pages 954, 1399, 5050, 33722

IBM i

Code page:

1399 Does not convert to code page 5039

5035 Does not convert to code page 5039

HP-UX

Code page:

932 Does not convert to code pages 942, 943, 1399

954 Does not convert to code pages 942, 943, 1399

5039 Does not convert to code pages 942, 943, 1399

HP Integrity NonStop Server

Code page:

943 Does not convert to code page 1399

5050 Does not convert to code page 1399

Japanese Kanji/ Katakana Mixed:

Details of CCSIDs and CCSID conversion for Japanese Kanji/Katakana Mixed.

The following table shows the native CCSIDs for Japanese Kanji/ Katakana Mixed on supported platforms:

Platform	Native CCSIDs
z/OS	1390, 5026 (see Note 1)
IBM i	5026 (see Note 1)
AIX	932, 5050, 33722 (see Note 2)
HP-UX	932, 954, 5039 (see Note 3)
Windows	932, 943 (see Notes 4 and 5)
Linux, HP Integrity NonStop Server, Solaris	943, 5050

Platform	Native CCSIDs

Note:

1. CCSID 1390 does not accept lower case characters. 5026 is a CCSID related to code page 930. CCSID 5026 is the CCSID reported on IBM i when the Japanese Katakana (DBCS) feature is selected.
2. 5050 and 33722 are CCSIDs related to base code page 954 on AIX. The CCSID reported by the operating system is 33722.
3. Code sets `japan15` and SJIS on HP-UX are represented by CCSID 932. These have a few DBCS characters having different representations in SJIS so 932 may be converted incorrectly if the conversion is not performed on an HP-UX system. WebSphere MQ for HP-UX supports 5039, the correct CCSID for HP SJIS. A change to file `/var/mqm/conv/ccsid.tbl` can be made to change the CCSID used from 932 to 5039.
4. Windows NT uses code page 932 but this is best represented by the CCSID of 943. However, not all platforms of WebSphere MQ support this CCSID.

   On WebSphere MQ for Windows, CCSID 932 is used to represent code page 932, but a change to file `../conv/table/ccsid.tbl` can be made that changes the CCSID used to 943.
5. WebSphere MQ does not support code pages based on the JIS X 0213 (JIS2004) standard.

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

1390 Does not convert to code pages 954, 5026, 5050, 33722

Does not accept lower case characters.

5026 Does not convert to code pages 954, 1390, 5050, 33722

IBM i

Code page:

5026 Does not convert to code pages 1390, 5039

HP-UX

Code page:

932 Does not convert to code pages 942, 943, 1390

954 Does not convert to code pages 942, 943, 1390

5039 Does not convert to code pages 942, 943, 1390

HP Integrity NonStop Server

Code page:

943 Does not convert to code page 1390

5050 Does not convert to code page 1390

Korean:

Details of CCSIDs and CCSID conversion for Korean.

The following table shows the native CCSIDs for Korean on supported platforms:

Platform	Native CCSIDs
z/OS, IBM i	933, 1364
AIX, HP-UX, Linux, HP Integrity NonStop Server, Solaris	970
Windows	949, 1363

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

933 Does not convert to code page 970

1364 Does not convert to code page 970

HP-UX

Code page:

970 Does not convert to code pages 949, 1363, 1364

Simplified Chinese:

Details of CCSIDs and CCSID conversion for Simplified Chinese.

The following table shows the native CCSIDs for Simplified Chinese on supported platforms:

Platform	Native CCSIDs
z/OS	935, 1388
IBM i	935, 1388
AIX	1383, 1386
HP-UX	1381 (see Note 1)
Windows	1381, 1386(see Note 2)
Linux, HP Integrity NonStop Server, Solaris	1383

Platform	Native CCSIDs
Note: 1. Code sets `prc15` and `hp15CN` on HP-UX are represented by CCSID 1381. 2. Windows uses code page 936 but this is best represented by the CCSID of 1386. However, not all platforms of WebSphere MQ support this CCSID. On WebSphere MQ for Windows CCSID 1381 is used to represent code page 936, but a change to file `../conv/table/ccsid.tbl` can be made which changes the CCSID used to 1386. 3. WebSphere MQ supports phase one of the Chinese GB18030 standard. On z/OS, Linux, Windows, and Solaris, conversion support is provided between Unicode (UTF-8 and UCS-2) and CCSID 1388 (EBCDIC with GB18030 extensions), Unicode (UTF-8 and UCS-2) and CCSID 5488 (GB18030 phase one), and between CCSID 1388 and CCSID 5488. **Note:** On IBM i, support is provided by the operating system for conversion between Unicode (UTF-8 and UCS-2) and CCSID 1388 (EBCDIC with GB18030 extensions). On HP-UX there is currently no support available on the HP11 operating system for GB18030. On HP11i, patch PHCO_26456 provides conversion support between GB18030 (CCSID 5488) and Unicode. Support is not provided for the conversion between GB18030 and 1388 (EBCDIC).	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

935 Does not convert to code page 1383

1388 Does not convert to code page 1383

HP-UX

Code page:

1381 Does not convert to code pages 1383, 1386, 1388

Traditional Chinese:

Details of CCSIDs and CCSID conversion for Traditional Chinese.

The following table shows the native CCSIDs for Traditional Chinese on supported platforms:

Platform	Native CCSIDs
z/OS, IBM i	937
HP-UX	938, 950, 964 (see Note)
Windows	950
AIX, HP Integrity NonStop Server, Solaris, Linux	950, 964
Note: Code set `roc15` on HP-UX is represented by CCSID 938.	

All platforms support conversion between their native CCSIDs and the native CCSIDs of other platforms, with the following exceptions.

z/OS

Code page:

937 Does not convert to code page 964

1388 Does not convert to code page 1383

HP-UX

Code page:

938 Does not convert to code page 948

950 Does not convert to code page 948

964 Does not convert to code page 948

Linux, Solaris

Code page:

964 Does not convert to code page 938

Unicode conversion support:

Some platforms support the conversion of user data to or from Unicode encoding. The two forms of Unicode encoding supported are UCS-2 (CCSIDs 1200, 13488, and 17584) and UTF-8 (CCSID 1208).

The term *UCS-2* is often used interchangeably but incorrectly with *UTF-16*. UCS-2 is a fixed-width encoding where each character occupies 2 bytes. UTF-16 is variable-width encoding that is a superset of UCS-2. In addition to the 2-byte UCS-2 characters, UTF-16 contains characters, known as surrogate pairs, that are 4 bytes in length. WebSphere MQ does not support surrogate pairs. The support for UTF-16 and UTF-8 in WebSphere MQ is therefore limited to those Unicode characters that can be encoded in UCS-2.

Note: WebSphere MQ does not support UCS-2 queue manager CCSIDs so message header data cannot be encoded in UCS-2.

WebSphere MQ AIX support for Unicode

On WebSphere MQ for AIX conversion to and from Unicode CCSIDs is supported for the CCSIDs in the following table.

037	273	278	280	284	285
297	423	437	500	813	819
850	852	856	857	858	860
861	865	867	869	875	878
880	901	902	912	915	916
920	923	924	932	933	935
937	938	939	942	943	948
949	950	954	964	970	1026
1046	1089	1129	1130	1131	1132
1133	1140	1141	1142	1143	1144
1145	1146	1147	1148	1149	1200
1153	1156	1157	1208	1250	1251
1253	1254	1258	1280	1281	1282
1283	1284	1285	1363	1364	1381
1383	1386	1388	4899	5026	5035
5050	5346	5347	5348	5349	5350

5351	5352	5353	5354	5488	9044
9048	9449	12712	13488	17584	33722

WebSphere MQ HP-UX support for Unicode

On WebSphere MQ for HP-UX conversion to and from Unicode CCSIDs is supported for the CCSIDs listed in the following table.

437	737	813	819	850	852
855	857	861	864	865	866
869	874	912	915	916	920
932	938	950	954	964	970
1051	1089	1140	1141	1142	1143
1144	1145	1146	1147	1148	1149
1200	1208	1250	1251	1252	1253
1254	1255	1256	1257	1258	1381
5050	5488	13488	33722		

WebSphere MQ for Windows, Solaris, and Linux support for Unicode

On WebSphere MQ for Windows, WebSphere MQ for Solaris, and WebSphere MQ for Linux conversion to, and from, Unicode CCSIDs is supported for the CCSIDs in the following table.

037	277	278	280	284	285
290	297	300	301	420	424
437	500	813	819	833	835
836	837	838	850	852	855
856	857	858	860	861	862
863	864	865	866	867	868
869	870	871	874	875	878
880	891	897	901	902	903
904	912	913 (5)	915	916	918
920	921	922	923	924	927
928	930	931 (1)	932 (2)	933	935
937	938 (3)	939	941	942	943
947	948	949	950	951	954 (4)
964	970	1006	1025	1026	1027
1040	1041	1042	1043	1046	1047
1051	1088	1089	1097	1098	1112
1114	1115	1122	1123	1124	1129
1130	1132	1133	1140	1141	1142
1143	1144	1145	1146	1147	1148
1149	1153	1156	1157	1200	1208
1250	1251	1252	1253	1254	1255
1256	1257	1258	1275	1280	1281
1282	1283	1363	1364	1380	1381
1383	1386	1388	4899	5050	5346
5347	5348	5349	5350	5351	5352
5353	5354	5488 (5)	9044	9048	9449
12712	13488	17584	33722 (4)		

Notes:

1. 931 uses 939 for conversion.
2. 932 uses 942 for conversion.
3. 938 uses 948 for conversion.
4. 954 and 33722 use 5050 for conversion.
5. On Windows, Linux, and Solaris only.

IBM i support for Unicode

For details on UNICODE support refer to the appropriate IBM i publication relating to your operating system.

WebSphere MQ for z/OS support for Unicode

On WebSphere MQ for z/OS conversion to and from the Unicode CCSIDs is supported for the following CCSIDs:

37	256	259	273	275	277
278	280	282	284	285	290
293	297	300	301	367	420
423	424	437	500	720	737
775	803	806	808	813	819
833	834	835	836	837	838
848	849	850	851	852	855
856	857	858	859	860	861
862	863	864	865	866	867
868	869	870	871	872	874
875	878	880	891	895	896
897	901	902	903	904	905
912	914	915	916	918	920
921	922	923	924	927	928
930	932	933	935	937	939
941	942	943	944	946	947
948	949	950	951	1004	1006
1008	1009	1010	1011	1012	1013
1014	1015	1016	1017	1018	1019
1025	1026	1027	1040	1041	1042
1043	1046	1047	1051	1088	1089
1097	1098	1112	1114	1115	1122
1123	1124	1125	1126	1129	1130
1131	1132	1133	1137	1140	1141
1142	1143	1144	1145	1146	1147
1148	1149	1153	1154	1155	1156
1157	1158	1159	1160	1161	1162
1164	1200	1208	1250	1251	1252
1253	1254	1255	1256	1257	1258
1275	1276	1277	1280	1281	1282
1283	1284	1285	1351	1362	1363
1364	1370	1371	1380	1381	1385
1386	1388	1390	1399	4899	4909
4930	4933	4948	4951	4952	4960
4971	5012	5039	5104	5123	5142
5210	5346	5347	5348	5349	5350
5351	5352	5353	5354	5488	8482

8612	9027	9030	9044	9048	9049
9056	9061	9066	9238	9449	12712
13121	13218	13488	16684	16804	17248
17584	21427	28709			

Coding standards on 64-bit platforms

Use this information to learn about coding standards on 64-bit platforms and the preferred data types.

Preferred data types

These types never change size and are available on both 32-bit and 64-bit WebSphere MQ platforms:

Name	Length
MQLONG	4 bytes
MQULONG	4 bytes
MQINT32	4 bytes
MQUINT32	4 bytes
MQINT64	8 bytes
MQUINT64	8 bytes

Standard data types:

Learn about standard data types on 32-bit UNIX, 64-bit UNIX, and 64-bit Windows applications.

32-bit UNIX applications

This section is included for comparison and is based on Solaris. Any differences with other UNIX platforms are noted:

Name	Length
char	1 byte
short	2 bytes
int	4 bytes
long	4 bytes
float	4 bytes
double	8 bytes
long double	16 bytes
	Note that on AIX and Linux PPC a long double is 8 bytes.
pointer	4 bytes
ptrdiff_t	4 bytes
size_t	4 bytes
time_t	4 bytes
clock_t	4 bytes
wchar_t	4 bytes
	Note that on AIX a wchar_t is 2 bytes.

64-bit UNIX applications

This section is based on Solaris. Any differences with other UNIX platforms are noted:

Name	Length
char	1 byte
short	2 bytes
int	4 bytes
long	8 bytes
float	4 bytes
double	8 bytes
long double	16 bytes
	Note that on AIX and Linux PPC a long double is 8 bytes.
pointer	8 bytes
ptrdiff_t	8 bytes
size_t	8 bytes
time_t	8 bytes
clock_t	8 bytes
	Note that on the other UNIX platforms a clock_t is 4 bytes.
wchar_t	4 bytes
	Note that on AIX a wchar_t is 2 bytes.

Windows 64-bit applications

Name	Length
char	1 byte
short	2 bytes
int	4 bytes
long	4 bytes
float	4 bytes
double	8 bytes
long double	8 bytes
pointer	8 bytes
	Note that all pointers are 8 bytes.
ptrdiff_t	8 bytes
size_t	8 bytes
time_t	8 bytes
clock_t	4 bytes
wchar_t	2 bytes
WORD	2 bytes
DWORD	4 bytes
HANDLE	8 bytes
HFILE	4 bytes

Coding considerations on Windows

HANDLE hf;

Use

```
hf = CreateFile((LPCTSTR) FileName,
                Access,
                ShareMode,
                xihSecAttsNTRestrict,
                Create,
                AttrAndFlags,
                NULL);
```

Do not use

```
HFILE hf;
hf = (HFILE) CreateFile((LPCTSTR) FileName,
                        Access,
                        ShareMode,
                        xihSecAttsNTRestrict,
                        Create,
                        AttrAndFlags,
                        NULL);
```

as this produces an error.

size_t len fgets

Use

```
size_t len
while (fgets(string1, (int) len, fp) != NULL)
len = strlen(buffer);
```

Do not use

```
int len;

while (fgets(string1, len, fp) != NULL)
len = strlen(buffer);
```

printf

Use

```
printf("My struc pointer: %p", pMyStruc);
```

Do not use

```
printf("My struc pointer: %x", pMyStruc);
```

If you need hexadecimal output, you have to print the upper and lower 4 bytes separately.

char *ptr

Use

```
char * ptr1;
char * ptr2;
size_t bufLen;

bufLen = ptr2 - ptr1;
```

Do not use

```
char *ptr1;
char *ptr2;
UINT32 bufLen;

bufLen = ptr2 - ptr1;
```

alignBytes

Use

```
alignBytes = (unsigned short) ((size_t) address % 16);
```

Do not use

```
void *address;
unsigned short alignBytes;

alignBytes = (unsigned short) ((UINT32) address % 16);
```

len

Use

```
len = (UINT32) ((char *) address2 - (char *) address1);
```

Do not use

```
void *address1;
void *address2;
UINT32 len;

len = (UINT32) ((char *) address2 - (char *) address1);
```

sscanf

Use

```
MQLONG SBCSprt;

sscanf(line, "%d", &SBCSprt);
```

Do not use

```
MQLONG SBCSprt;

sscanf(line, "%ld", &SBCSprt);
```

`%ld` tries to put an 8-byte type into a 4-byte type; only use `%l` if you are dealing with an actual `long` data type. MQLONG, UINT32 and INT32 are defined to be four bytes, the same as an `int` on all WebSphere MQ platforms:

SOAP reference

WebSphere MQ transport for SOAP reference information arranged alphabetically.

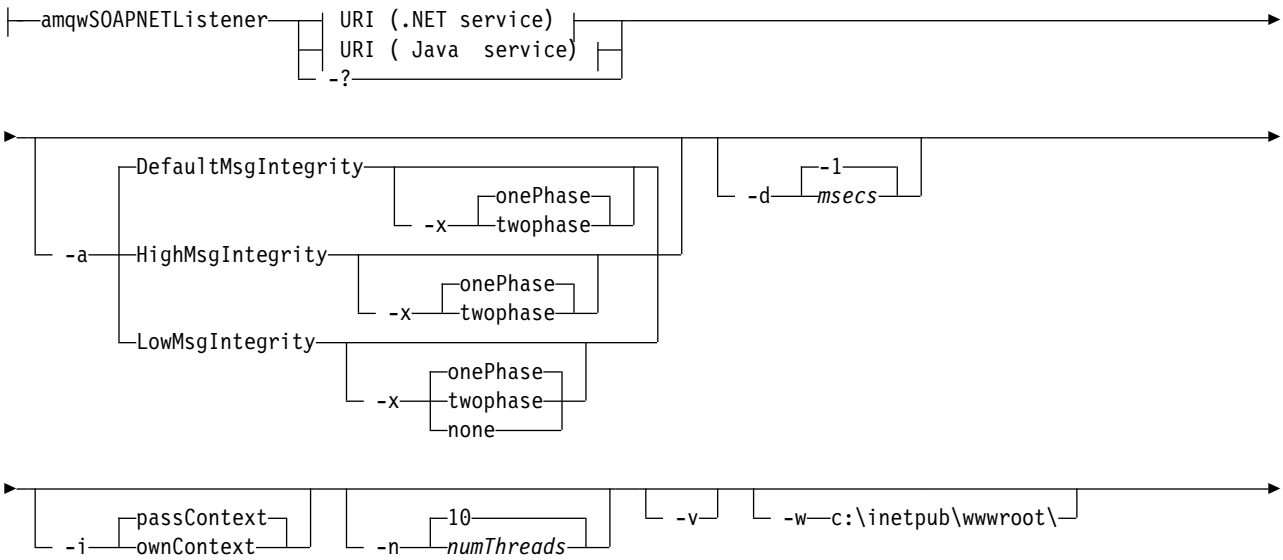
amqwSOAPNETListener: WebSphere MQ SOAP listener for .NET Framework 1 or 2

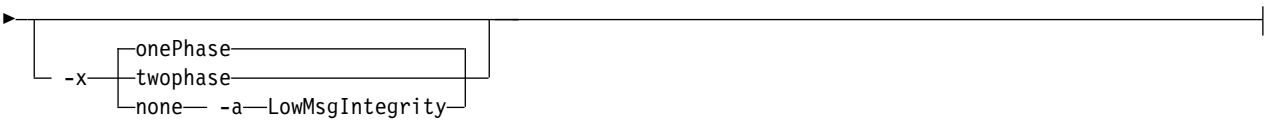
Syntax and parameters for the WebSphere MQ SOAP listener for .NET Framework 1 or 2.

Purpose

Starts the WebSphere MQ SOAP listener for .NET Framework 1 or 2.

.NET:





Required parameters

URI *platform*
See "URI syntax and parameters for Web service deployment" on page 2307.

-? Print out help text describing how the command is used.

Optional parameters

-a *integrityOption*
integrityOption specifies the behavior of WebSphere MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

DefaultMsgIntegrity
For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

LowMsgIntegrity
For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

HighMsgIntegrity
For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

-d *msecs*
msecs specifies the number of milliseconds for the WebSphere MQ SOAP listener to stay alive if request messages have been received on any thread. If *msecs* is set to -1, the listener stays alive indefinitely.

-i *Context*
Context specifies whether the listeners pass identity context. *Context* takes the following values:

passContext
Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

ownContext
The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

-n *numThreads*
numThreads specifies the number of threads in the generated startup scripts for the WebSphere MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

-v -v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

-w *serviceDirectory*
serviceDirectory is the directory containing the web service.

-x *transactionality*
transactionality specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

onePhase
WebSphere MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. WebSphere MQ transactions ensure that the response messages are written exactly once.

twoPhase
Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

none No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

.NET Example

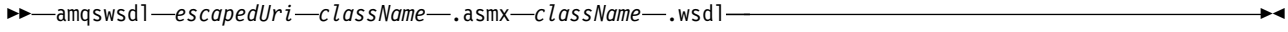
```
amqwSOAPNETlistener
-u "jms:/queue?destination=myQ&connectionFactory=()
&targetService=myService&initialContextFactory=com.ibm.mq.jms.Nojndi"
-w C:/wmqsoap/demos
-n 20
```

amqswsdl: generate WSDL for .NET Framework 1 or 2 service

amqswsdl takes a Web service written for .NET Framework 1 or 2, and generates the WSDL for the class, inserting the URI you provide for the WebSphere MQ transport for SOAP into the generated WSDL.

Purpose

Use **amqswsdl** to generate WSDL containing the URI of the service deployed to WebSphere MQ. Use the WSDL to generate client proxies.



Parameters

***escapedUri* (Input)**
The URI of the service, with all "&" escaped to "&amp.". For example:

```
"jms:/queue?destination=REQUESTDOTNET
&amp.initialContextFactory=com.ibm.mq.jms.Nojndi
&amp.connectionFactory=(connectQueueManager(QM1)binding(server))
&amp.targetService=Quote.asmx"
```

***className*.asmx (Input)**
The service class.

***className*.wsdl (Output)**
The service WSDL.

Description

If the class is implemented using the code-behind programming model, you must build *className*.dll and store it in ./bin.

amqwclientconfig: create Axis 1.4 Web services client deployment descriptor for WebSphere MQ transport for SOAP

amqwclientconfig creates the client-config.wsdd Axis 1.4 client deployment descriptor file.

Purpose

It adds the jms:/ transport to the descriptor, and registers java:com.ibm.mq.soap.transport.jms.WMQSender as the class to handle SOAP requests for the jms: transport.

Syntax

Description

amqwclientconfig calls **amqwsetcp** to set the CLASSPATH and runs the command:

```
java org.apache.axis.utils.Admin client "%WMQSOAP_HOME%\bin\amqwclientTransport.wsdd"
```

amqwdeployWMQService: deploy Web service utility

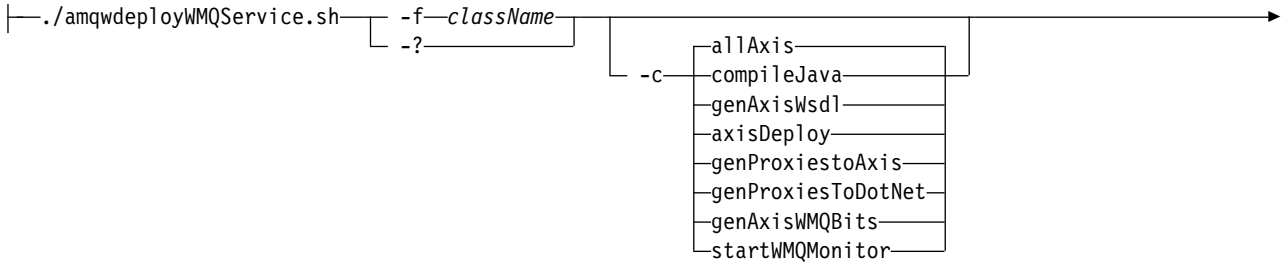
The deployment utility prepares a service class for use as a Web service using WebSphere MQ as the transport.

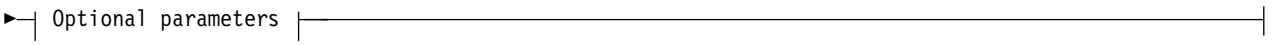
Purpose

Use the deployment utility to generate the files that are needed to deploy an Axis 1.4, .NET Framework 1 or .NET Framework 2 service. Use the files to deploy a service invoked by WebSphere MQ. The files generated by **amqwdeployWMQService** are shown in "Output files from **amqwdeployWMQService**" on page 2281.

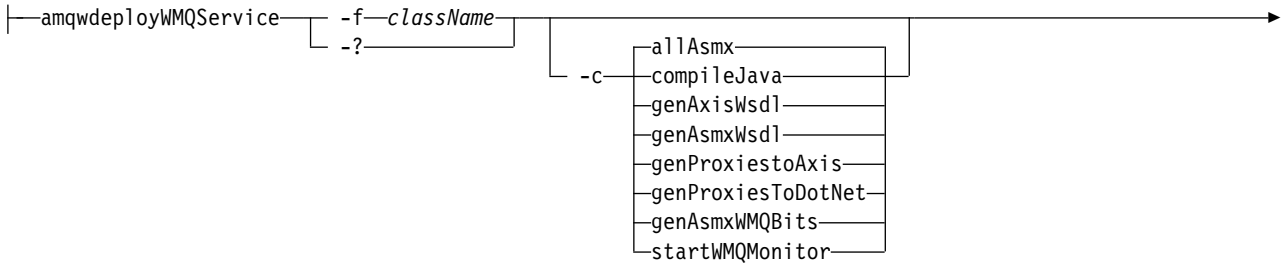
Syntax diagram

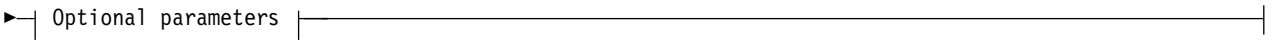
UNIX and Linux systems:



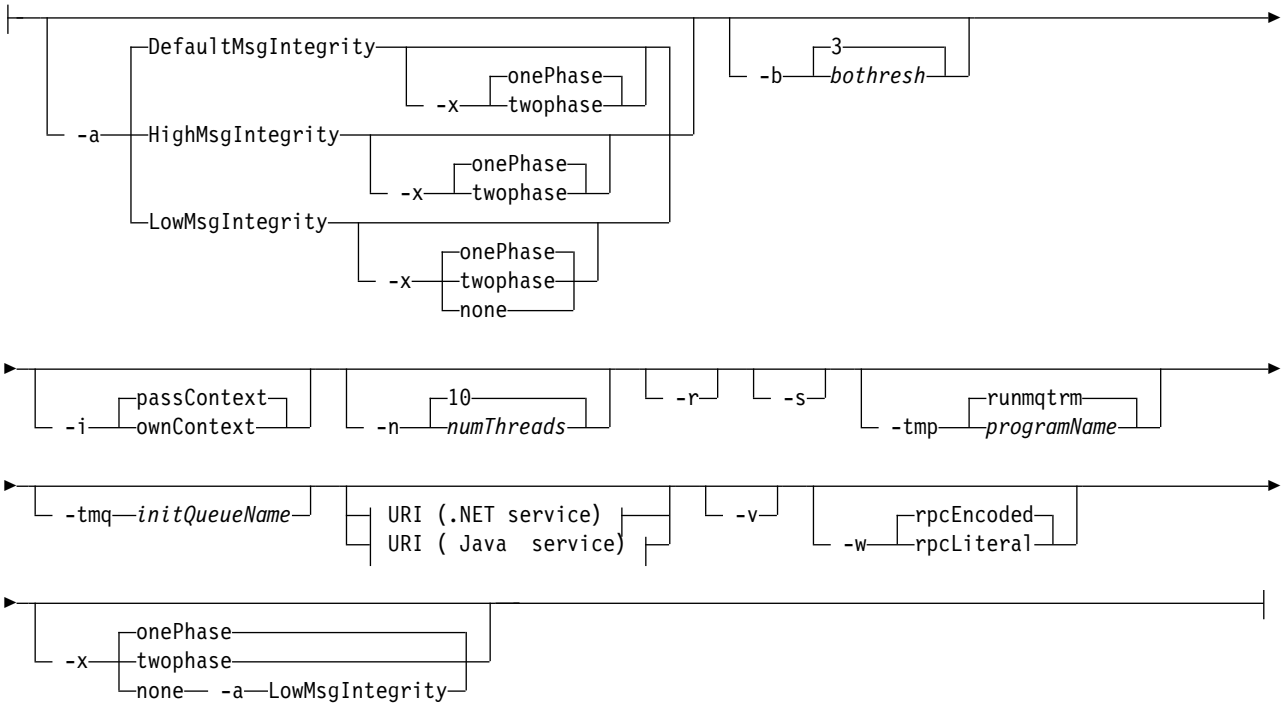


Windows:





Optional parameters:



Required parameters

-f *className*

className is the name of the class to be deployed. For Axis services *className* is the Java source file, and for .NET services, the `.asmx` file. Figure 20 on page 2278 illustrates the deployment of an Axis service and Figure 21 on page 2278 of a .NET service.

```
amqwdeployWMQService -f javaDemos/service/StockQuoteAxis.java
```

Figure 20. Example deployment of Axis service

```
amqwdeployWMQService -f StockQuoteDotNet.asmx
```

Figure 21. Example deployment of .NET service

For Java, *className* must be fully qualified by the package name. It can be specified as a path name with directory separators or as a class name with period separators. The generated class is located in `./generated/client/remote/path name`. For a .NET service, although the directory can be specified, generated Java proxies are always located in `./generated/client/remote/dotNetService`.

If you specify a URI with the -u option and within the URI specify *targetService*, the deployment utility checks the *className*. *className* must match *targetService*. If the class and service do not match, the deployment utility displays an error message and exits.

-? Print out help text describing how the command is used.

Optional parameters

-a *integrityOption*
integrityOption specifies the behavior of WebSphere MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

DefaultMsgIntegrity
For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

LowMsgIntegrity
For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

HighMsgIntegrity
For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

-b *bothresh*
bothresh specifies the backout threshold setting for the request queue. The default is 3.

-c *operation*
operation specifies which part of the deployment process to execute. *operation* is one of the following options:

allAxis
Perform all compile and setup steps for an Axis or Java service[3].

compileJava
Compile the Java service: `.java` to `.class`.

genAxisWsdl
Generate WSDL: `.class` to `.wsdl`.

3. Default if *className* has a .java extension

axisDeploy
Deploy the class file: `.wsdl` to `.wsdd`, apply `.wsdd`.

genProxiestoAxis
Generate proxies: `.wsdl` to `.java` and `.class`.

genAxisWMQBits
Set up WebSphere MQ queues, WebSphere MQ SOAP listeners and triggers for an Axis service.

allAsmx
Perform all setup steps for a .NET service[4].

genAsmxWsdl
Generate WSDL: `.asmx` to `.wsdl`.

genProxiesToDotNet
Generate proxies: `.wsdl` to `.java`, `.class`, `.cs` and `.vb`.

genAsmxWMQBits
Set up WebSphere MQ queues, WebSphere MQ SOAP listeners and triggers

startWMQMonitor
Start the trigger monitor for WebSphere MQ SOAP services.

Note: **runmqtrm** runs under the `mqm` user ID. If security is an issue, you must ensure that the listeners are started under appropriate user IDs.

-i *Context*
Context specifies whether the listeners pass identity context. *Context* takes the following values:

passContext
Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

ownContext
The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

-n *numThreads*
numThreads specifies the number of threads in the generated startup scripts for the WebSphere MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

-r *-r* specifies that any existing request or trigger monitor queue definitions are replaced. Trigger monitor queues are replaced only if -tmq is also specified. Queues are re-created with standard default attributes and existing messages on the queues are deleted. If the -r option is not used then any existing queue definitions are not altered and existing messages are not deleted. By not specifying -r, you ensure that any customized queue attributes are preserved.

4. Default if *className* has a .asmx extension.

-s Configure the listener to run as a WebSphere MQ service. If -s and -tmq are both specified, the deployment utility displays an error message and exits.

-tmp *programName*
programName specifies the name of a trigger monitor program. Use **-tmp** *programName* in a UNIX or Linux environment as an alternative to using **runmqtrm**. Programs it initiates run under `mqm` authority.

For example:

```
amqwdeployWMQService -f javaDemos/service/StockQuoteAxis.java
-tmq trigger.monitor.queue -tmp trigmon
```

-tmq *queueName*
queueName specifies a trigger monitor queue name. WebSphere MQ process definitions are created to configure automatic triggering of WebSphere MQ SOAP listeners with the associated trigger monitor queue name. If the option is not specified then no triggering configuration is defined by the deployment utility. If -s and -tmq are both specified, the deployment utility displays an error message and exits.

URI *platform*
See "URI syntax and parameters for Web service deployment" on page 2307.

-v -v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

-w -w controls the style of WSDL to generate. The default is rpcEncloded, for compatibility with previous releases of WebSphere MQ transport for SOAP. Use rpcLiteral to create WSDL compatible with Axis2 client proxy generation. rpcEncoded is not compatible with WS-I recommendations.

-x *transactionality*
transactionality specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

onePhase
WebSphere MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. WebSphere MQ transactions ensure that the response messages are written exactly once.

twoPhase
Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

none No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

Errors

On Windows, if errors are reported from **amqswsdl**, try issuing the following command to register `.asmx` files as services.

```
%windir%/Microsoft.NET/Framework/version number/aspnet_regiis.exe -ir
```

The problem typically occurs on systems where IIS has not been installed, or IIS has been installed after NET. The problem is encountered when **amqswsdl** generates the `.wsdl` files.

Note: The registry keys are also required to permit the listener to invoke the services. If you use your own customized deployment procedures, you might not encounter the problem until run time.

Output files from amqwdeployWMQService:

A list of the directories and files output from **amqwdeployWMQService**

Table 229. Output files from **amqwdeployWMQService**

Outputs	Description	Output directory	Filename
.class	Compiled Java source file	./generated/server/*server package*	*classname*.class
.wsdl	Service description	./generated	*className*Axis_Wmq.wsdl *className*DotNet_Wmq.wsdl
.wsdd	Axis client and service deployment files	./	client-config.wsdd server-config.wsdd
		./generated/server/*server package*	*className*_deploy.wsdd *className*_undeploy.wsdd
Client source (.vb, .cs, .java)	.Net client stubs to Axis service	./generated/client	*classname*AxisService.cs *classname*AxisService.vb
	.Net client stubs to .Net service	./generated/client	*classname*DotNet.cs *classname*DotNet.vb
Client helper (.java and .class)	Java client proxies to .Net service	./generated/server/soap/client/remote/dotnetService	*className*DotNet.class *className*DotNet.java *className*DotNetLocator.class *className*DotNetLocator.java *className*DotNetSoap12Stub.class *className*DotNetSoap12Stub.java *className*DotNetSoap_BindingStub.class *className*DotNetSoap_BindingStub.java *className*DotNetSoap_PortType.class *className*DotNetSoap_PortType.java
	Java client proxies to Axis service	./generated/server/soap/client/remote/*client package*	SoapServer*className*AxisBindingSoapStub.class SoapServer*className*AxisBindingSoapStub.java *className*Axis.class *className*Axis.java *className*AxisService.class *className*AxisService.java *className*AxisServiceLocator.class *className*AxisServiceLocator.java
Scripts (.cmd and .sh)	Listener scripts	/generated/server	startWMQJListener.cmd startWMQJListener.sh startWMQNListener.cmd endWMQJListener.cmd endWMQJListener.sh endWMQNListener.cmd

Usage notes for amqwdeployWMQService:

Describes the tasks performed by **amqwdeployWMQService**.

The deployment utility performs the following actions.

1. Checks paths to the following files:
   - axis.jar.
   - *WMQSOAP_HOME*/java/lib/com.ibm.mq.soap.jar.
   - On Windows, csc.exe
2. On Windows, uses either %SystemRoot%\Microsoft.NET\Framework\v1.1.432 or, if the C# compiler is installed, the path to csc.exe as the path to the .NET Framework.

   Note: If you have Microsoft Visual Studio 2008 installed (Version 9), wsdl.exe is not in the path to csc.exe. You need to add the path to the .NET framework to your Path variable; for example:
   ```
   Set Path=C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727;%Path%
   ```
3. Creates the ./generated directory, and required subdirectories, if they do not exist.
4. For Java services, compiles the source into *className*.class.
5. Generates WSDL.
6. For Java services, creates deployment descriptor files *className*_deploy.wsdd and *className*_undeploy.wsdd
7. For Java services, creates or updates the Axis deployment descriptor file, server-config.wsdd.
8. Generates the client proxies for Java, C# and Visual Basic from the WSDL.

   Note: On Windows, the deploy utility generates proxies for Visual Basic and C# regardless of the language in which the service is written. The WSDL and the proxies generated from it include the appropriate URI to call the service:

   a.

```
jms:/queue?destination=SOAPN.demos@WMQSOAP.DEMO.QM
&connectionFactory=(connectQueueManager(WMQSOAP.DEMO.QM))
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=StockQuoteDotNet.asmx
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

Figure 22. Example URI in generated .NET client to call .NET service

   b.

```
jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM
&connectionFactory=(connectQueueManager(WMQSOAP.DEMO.QM))
&initialContextFactory=com.ibm.mq.jms.Nojndi
&targetService=soap.server.StockQuoteAxis.java
&replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
```

Figure 23. Example URI in generated .NET client to call Axis 1 service

9. Compiles the Java proxies.
10. Creates a WebSphere MQ queue, *requestQueue* to hold requests for the service. The default queue name is of the form SOAPJ.*directory*, or you can specify *requestQueue* in the -u URI option.
11. Creates command and shell script files to start the WebSphere MQ SOAP listeners that process the request queue.
12. If the -tmq option has been used, the deployment utility creates WebSphere MQ definitions to trigger WebSphere MQ SOAP listener processes automatically.

- The deployment utility uses the APPLICID attribute of the **runmqsc** DEFINE PROCESS command to contain a command to start the listener. The command has the name of the deployment directory embedded in it. The APPLICID field has a maximum length of 256, which limits the maximum length of the deployment directory. The directory limit for Java services is as follows:
  - UNIX and Linux systems: 218
  - Windows: 197 minus the length of the request queue name.

  For .NET services, the directory limit is as follows:
  - Windows: 209 minus length of the service name, minus the .asmx extension.
- The deployment utility checks whether the limit for APPLICID is exceeded. If the limit is exceeded, the utility does not attempt to define the triggering process. It displays an error message and the deployment process fails without performing any deployment steps.

The following examples show the configuration and start commands generated by the deployment utility to start a WebSphere MQ SOAP listener.

```
DEFINE PROCESS(requestQueue) APPLICID(applicIDStr) REPLACE
ALTER  QLOCAL (requestQueue) TRIGTYPE(FIRST) TRIGGER
       PROCESS(requestQueue) INITQ(initQueueName) TRIGMPRI(0)
```

Figure 24. WebSphere MQ configuration commands to trigger a SOAP listener.

```
applicIDStr = start "Java WMQSoapListener -requestQueue"
                    /min .\generated\server\startWMQJListener.cmd;
```

Figure 25. Starting Axis SOAP listener on Windows

```
applicIDStr = start "WMQAsmxListener -className\
                    /min .\generated\server\startWMQNListener.cmd;
```

Figure 26. Starting .NET SOAP listener on Windows

```
applicIDStr = xterm -iconic -T \"Java WMQSoapListener_requestQueue\"
                   -e ./generated/server/startWMQJListener.sh & #
```

Figure 27. Starting Axis SOAP listener on UNIX and Linux systems

amqwRegisterdotNet: register WebSphere MQ transport for SOAP to .NET

Register WebSphere MQ transport for SOAP to the global assembly cache on .NET.

Purpose

amqwRegisterdotNet registers the WebSphere MQ SOAP sender, SOAP listener, and WSDL processor with .NET Framework 1 or 2.

Syntax

Description

amqwRegisterdotNet is run automatically during installation. You do not need to run it again if the .NET Framework you are using was installed before WebSphere MQ transport for SOAP. You can run it as many times as you want. Use it to reregister WebSphere MQ transport for SOAP with different .NET Framework versions.

Note: On Windows 2003 Server, you must also run the **aspnet_regiis** utility, even if you are not deploying to Internet Information Server (IIS). The location of the aspnet_regiis.exe utility might vary

with different versions of the Microsoft .NET Framework, but it is typically located in: `%SystemRoot%/Microsoft.NET/Framework/version number/aspnet_regiis`. If multiple versions are installed, use **aspnet_regiis** for the version of .NET Framework you are using.

Apache software license

Apache License Version 2.0, January 2004 http://www.apache.org/licenses/

http://www.apache.org/licenses/

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of

MQMD SOAP settings

The IBM WebSphere MQ SOAP sender and IBM WebSphere MQ SOAP listener create a message descriptor (**MQMD**). The topic describes the fields you must set in the MQMD if you create your own SOAP sender or listener.

Purpose

The values set in the **MQMD** control the exchange of messages between the IBM WebSphere MQ SOAP sender, the IBM WebSphere MQ SOAP listener, and the SOAP client program. If you create your own SOAP sender or listener, follow the rules in Table 230.

Description

Table 230 describes how the **MQMD** fields are set by the IBM WebSphere MQ SOAP sender and IBM WebSphere MQ SOAP listener. If you write your own sender or listener you must set these fields in accordance with the rules for exchanging messages. The IBM WebSphere MQ SOAP listener conforms to typical IBM WebSphere MQ message exchange protocols. If you write your own sender to work with the IBM WebSphere MQ SOAP listeners, you can set different **MQMD** values.

In Table 230, the values in the Setting column are organized as follows:

Request, One way
Settings made by IBM WebSphere MQ SOAP sender.

Response, Report
Settings made by IBM WebSphere MQ SOAP listener in response to IBM WebSphere MQ SOAP sender request.

ALL Settings made by both the IBM WebSphere MQ SOAP sender and IBM WebSphere MQ SOAP listener.

Customized sender
You can write your own sender. Typically, a customized sender overrides the standard report options.

Table 230. MQMD SOAP settings

Field name	Setting	Values
StrucId	**ALL** MQMD_STRUC_ID	'MD¬¬' [1]
Version	**ALL** MQMD_VERSION_2	2
Report	**ALL** MQRO_NONE + MQRO_NEW_MSG_ID + MQRO_COPY_MSG_ID_TO_CORREL_ID + MQRO_EXCEPTION + MQRO_EXPIRY + MQRO_DISCARD **Customized sender** See "Customized report options" on page 2290	52428800

Table 230. MQMD SOAP settings (continued)

Field name	Setting	Values
MsgType	**Request** MQMT_REQUEST **Response** MQMT_REPLY **Report** MQMT_REPORT **One way** MQMT_DATAGRAM	**MQMT_REQUEST** 1 **MQMT_REPLY** 2 **MQMT_REPORT** 4 **MQMT_DATAGRAM** 8
Expiry	**Request, One way** Specified by Expiry option in URI. Defaults to MQEI_UNLIMITED. **Response** Value of Expiry in request message **Report** MQEI_UNLIMITED	**MQEI_UNLIMITED** -1
Feedback	**Request, Response, One way** MQFB_NONE. **Report** • Generated by queue manager - value set according to normal rules. • Generated by IBM WebSphere MQ SOAP Listener: **MQRC_BACKOUT_THRESHOLD_REACHED** Backout threshold for multiple attempts exceeded. **MQRCCF_MD_FORMAT_ERROR** The message is not recognized as having an MQRFH2 header. **MQRC_RFH_PARM_MISSING** A required parameter, for example, SoapAction, in **MQRFH2** is missing. **MQRC_RFH_FORMAT_ERROR** A basic integrity check of the **MQRFH2** failed, for example, internal lengths are corrupted. **MQRC_RFH_ERROR** The **MQRFH2** passed an integrity check, but the body of the message is not set to MQFMT_NONE.	**MQFB_NONE** 0 **MQRC_BACKOUT_THRESHOLD_REACHED** 2362 **MQRCCF_MD_FORMAT_ERROR** 3023 **MQRC_RFH_PARM_MISSING** 2339 **MQRC_RFH_FORMAT_ERROR** 2421 **MQRC_RFH_ERROR** 2334
Encoding	**ALL** MQENC_NATIVE	Depends on environment
CodedCharSetId	**ALL** Set to UTF-8	1208

Table 230. MQMD SOAP settings (continued)

Field name	Setting	Values
Format	**Request, Response, One way** MQFMT_RF_HEADER_2 **Report** **Queue manager reports** Follows IBM WebSphere MQ rules **IBM WebSphere MQ SOAP listener reports** Format of the original request message.	**MQFMT_RF_HEADER_2** "MQRFH2 "
Priority	**Request, One way** Specified by Priority option in URI. Defaults to MQPRI_PRIORITY_AS_Q_DEF. **Response, Report** Value of Priority in the request message.	**MQPRI_PRIORITY_AS_Q_DEF** -1
Persistence	**Request, One way** MQPER_PERSISTENCE_AS_Q_DEF. **Response, Report** Value of Persistence in the request message.	**MQPER_PERSISTENCE_AS_Q_DEF** 2
MsgId	**Request, One way** Generated by the queue manager. **Response, Report** The IBM WebSphere MQ SOAP sender sets MQRO_NEW_MSG_ID and *MsgId* is generated	**Generated** Unique value generated by the queue manager
CorrelId	**Request, One way, Report** MQCI_NONE **Response, Report** The IBM WebSphere MQ SOAP sender sets MQRO_COPY_MSG_ID_TO_CORREL_ID and the listener copies *MsgId* from the request message.	**MQCI_NONE** 0
BackoutCount	**ALL** Not used	0
ReplyToQ	**Request** Specified by replyDestination option in URI. Defaults to SYSTEM.SOAP.RESPONSE.QUEUE. **Response, One way, Report** Left blank	
ReplyToQMgr	**ALL** Field left blank	Generated by the queue manager; see Reply-to queue and queue manager .

Table 230. MQMD SOAP settings (continued)

Field name	Setting	Values
UserIdentifier	**Request, One way, Report** Left blank **Response** Depends on the -i *passContext* option supplied to the listener, and the authority the listener is running under.	**Request, One way, Report** Generated by the queue manager; see "UserIdentifier (MQCHAR12)" on page 1752 **Response** *Variable*
AccountingToken	**ALL** MQACT_NONE	**MQACT_NONE** Null string or blanks Set by the queue manager; see "AccountingToken (MQBYTE32)" on page 1708
ApplIdentityData	**ALL** None	Null string or blanks [2]
PutApplType	**ALL** MQAT_NO_CONTEXT	**MQAT_NO_CONTEXT** 0 Value generated by queue manager; see "PutApplType (MQLONG)" on page 1738.
PutApplName	**ALL** None	Value generated by queue manager; see "PutApplName (MQCHAR28)" on page 1736.
PutDate	**ALL** None	Value generated by queue manager; see "PutDate (MQCHAR8)" on page 1740.
PutTime	**ALL** None	Value generated by queue manager; see "PutTime (MQCHAR8)" on page 1740.
ApplOriginData	**ALL** None	Null string or blanks [2]
GroupId	**Request, One way, Report** MQGI_NONE **Response** Field is copied from the request message	Nulls
MsgSeqNumber	**Request, One way, Report** Not used **Response** Field is copied from the request message	Generated by the queue manager; see Physical order on a queue.
Offset	**Request, One way, Report** Not used **Response** Field is copied from the request message	0
MsgFlags	**Request, One way, Report** MQMF_NONE **Response** Field is copied from the request message	**MQMF_NONE** 0 See "MsgFlags (MQLONG)" on page 1726

Table 230. MQMD SOAP settings (continued)

Field name	Setting	Values
OriginalLength	**Request, One way, Response** MQOL_UNDEFINED **Report** Length of original request message	**MQOL_UNDEFINED** -1
Notes: 1. The symbol ¬ represents a single blank character. 2. The value Null string or blanks denotes the null string in C, and blank characters in other programming languages.		

Customized report options

You can write your own SOAP sender and use it with the supplied listeners. Typically you might write a sender to change the choice of report options. The IBM WebSphere MQ SOAP listeners support most combinations of report options, as described in the following lists.

- Report options supported by IBM WebSphere MQ SOAP listeners:
  - MQRO_EXCEPTION
  - MQRO_EXCEPTION_WITH_DATA
  - MQRO_EXCEPTION_WITH_FULL_DATA
  - MQRO_DEAD_LETTER_Q
  - MQRO_DISCARD_MSG
  - MQRO_NONE
  - MQRO_NEW_MSG_ID
  - MQRO_PASS_MSG_ID
  - MQRO_COPY_MSG_ID_TO_CORREL_ID
  - MQRO_PASS_CORREL_ID
- Report options supported by the queue manager:
  - MQRO_COA
  - MQRO_COA_WITH_DATA
  - MQRO_COA_WITH_FULL_DATA
  - MQRO_COD
  - MQRO_COD_WITH_DATA
  - MQRO_COD_WITH_FULL_DATA
  - MQRO_EXPIRATION
  - MQRO_EXPIRATION_WITH_DATA
  - MQRO_EXPIRATION_WITH_FULL_DATA
- The following report options are not supported by the IBM WebSphere MQ SOAP listeners.
  - MQRO_PAN
  - MQRO_NAN

The behavior of IBM WebSphere MQ SOAP listeners in response to combinations of MQRO_EXCEPTION_* and MQRO_DISCARD is described in Table 231 on page 2291.

The notation MQRO_EXCEPTION_* indicates the use of either MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA or MQRO_EXCEPTION_WITH_FULL_DATA.

Table 231. Listener behavior resulting from MQRO_EXCEPTION_* and MQRO_DISCARD settings

	MQRO_DISCARD enabled	**MQRO_DISCARD not enabled**
MQRO_EXCEPTION_* enabled	Default behavior. Report messages are automatically generated if necessary and the original request discarded. If a report message could not be returned to the response queue the report message is sent to the dead letter queue.	Report messages are automatically generated if necessary and the original message is sent to the dead letter queue. If the report message could not be returned to the response queue it is also be sent to the dead-letter queue. In this case there are therefore two dead letter queue entries for the failed request.
MQRO_EXCEPTION_* not enabled	Report messages are not automatically generated when the incoming format is not recognized or the number of backout attempts is exceeded. The message is not sent to the dead letter queue. No notification is returned that the client can inspect, and the original request message is lost.	Report messages are not automatically generated when the incoming format is not recognized or the number of backout attempts is exceeded. The original request message is however written to the dead letter queue when a report would otherwise have been generated.

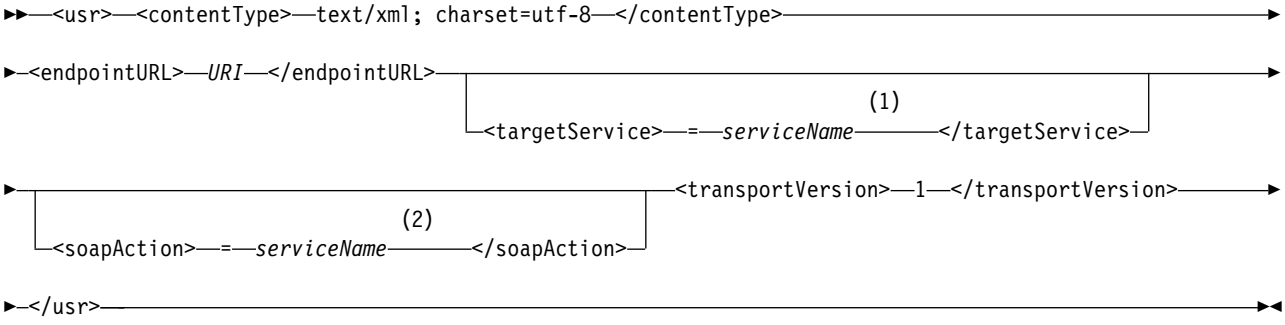
MQRFH2 SOAP settings

The WebSphere MQ SOAP senders and listeners create or expect to receive an MQRFH2 with the following settings.

Purpose

The WebSphere MQ SOAP senders add properties to the <usr> folder created by WebSphere MQ JMS. The properties contain information required by the SOAP container in the target environment. "Property syntax" describes the syntax of the properties when they are added to an MQRFH2. For the description an MQRFH2 header, see MQRFH2 - Rules and formatting header 2.

Property syntax



Notes:

1 targetService is required for .NET Framework 1 or 2, and not used on Axis 1.4.

2 soapAction is optional for .NET Framework 1 or 2, and not used on Axis 1.4.

Parameters

contentType

contentType always contains the string text/xml; charset=utf-8.

endpointURL
See "URI syntax and parameters for Web service deployment" on page 2307.

targetService
[5]On Axis, *serviceName* is the fully qualified name of a Java service, for example: `targetService=javaDemos.service.StockQuoteAxis`. If targetService is not specified, a service is loaded using the default Axis mechanism.

[6]On .NET, *serviceName* is the name of a .NET service located in the deployment directory, for example: `targetService=myService.asmx`. In the .NET environment, the targetService parameter makes it possible for a single WebSphere MQ SOAP listener to be able to process requests for multiple services. These services must be deployed from the same directory.

soapAction

transportVersion
transportVersion is always set to 1.

Example

The example shows an MQRFH2 and the following SOAP message. The lengths of the folders are shown in decimal.

Note: & in the URI is encoded as `&`

```
52464820 00000002 000002B0 00000001 RFH 0002 1208 0001
000004B8 20202020 20202020 00000000 1208 0000
000004B8                           1208
32  <mcd>
        <Msd>jms_bytes</Msd>
    </mcd>?
208 <jms>
        <Dst>queue://queue://SOAPJ.demos</Dst>
        <Rto>queue://WMQSOAP.DEMO.QM/SYSTEM.SOAP.RESPONSE.QUEUE</Rto>
        <Tms>1157388516465</Tms>
        <Cid>ID:000000000000000000000000000000000000000000000000</Cid>
        <Dlv>1</Dlv>
    </jms>
400 <usr>
        <contentType>text/xml; charset=utf-8</contentType>
        <transportVersion>1</transportVersion>
        <endpointURL>
            jms:/queue?destination=SOAPJ.demos@WMQSOAP.DEMO.QM
            &amp;connectionFactory=connectQueueManager(WMQSOAP.DEMO.QM)
                clientConnection(localhost%25289414%2529)
                clientChannel(TESTCHANNEL)
            &amp;replyDestination=SYSTEM.SOAP.RESPONSE.QUEUE
            &amp;initialContextFactory=com.ibm.mq.jms.Nojndi
        </endpointURL>
    </usr>
<?xml version="1.0" encoding="UTF-8"?>
    <soapenv:Envelope
            xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Body>
            <ns1:getQuote
                    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                    xmlns:ns1="soap.server.StockQuoteAxis_Wmq">
```

5. Java service only

6. .NET service only

```
            <in0 xsi:type="xsd:string">XXX</in0>
         </ns1:getQuote>
      </soapenv:Body>
   </soapenv:Envelope>
```

runivt: WebSphere MQ transport for SOAP installation verification test

An installation verification test suite (IVT) is provided with WebSphere MQ transport for SOAP. **runivt** runs a number of demonstration applications and ensures that the environment is correctly set up after installation.

Purpose

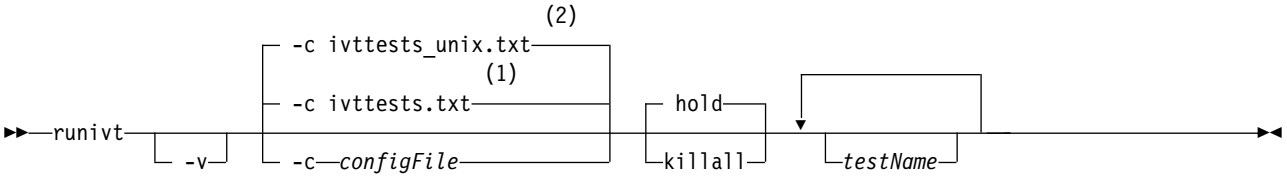
The **runivt** command uses the sample programs provided with the WebSphere MQ transport for SOAP to send Web service requests from clients to services. It runs tests for Axis 1.4, .NET Framework 1, and .NET Framework 2. The tests are configured in a test script file. The default test script file for Windows runs a combination of tests between Java and .NET clients and services.

Description

runivt must be run from its own directory.

The command starts listeners in a different command window. For this reason, you must run the command from an X Window System session on UNIX and Linux systems.

runivt syntax



Notes:

1 Default on Windows

2 Default on UNIX and Linux systems

runivt parameters

-v Verbose mode. Write fuller error messages to the console.

-c *configFile*
A configuration file defining the tests to be run. The default configuration file supplied with Windows, UNIX or Linux systems is used by default.

hold
Leave the listener running after the tests complete

killall
End the listener when the tests complete

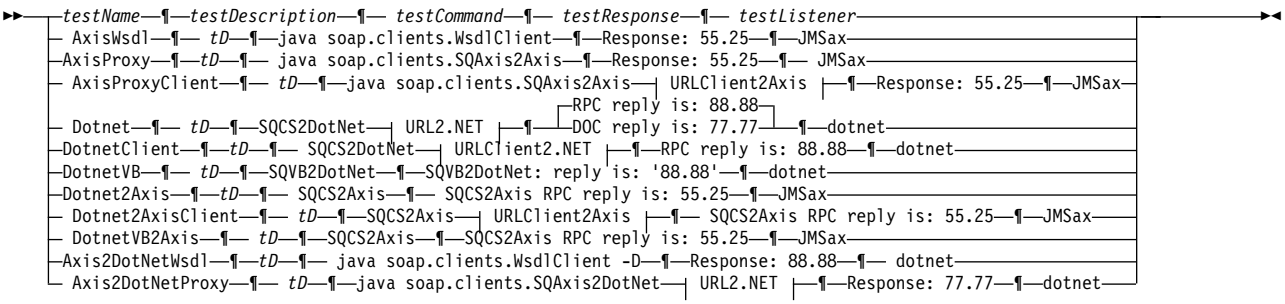
testName
A space separated list of the tests to run. The test names are selected from the configuration file. If no names are specified then all the tests in the configuration file are run.

Configuration file

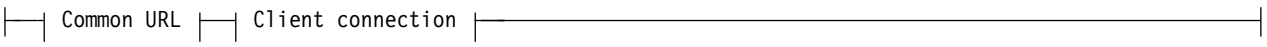
Each configuration file parameter is a separate line of the file. Leave a blank line between each group of parameters.

The parameters in the `ivttests.txt` parameter file are listed.

configFile syntax



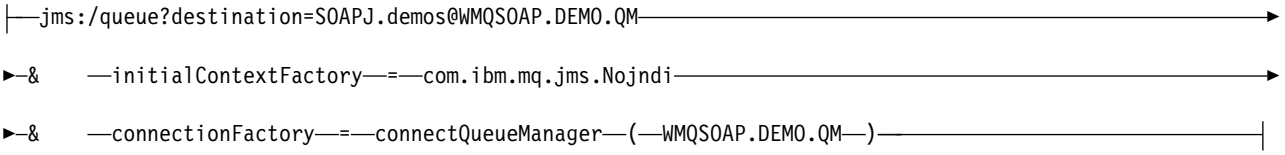
URLClient2Axis:



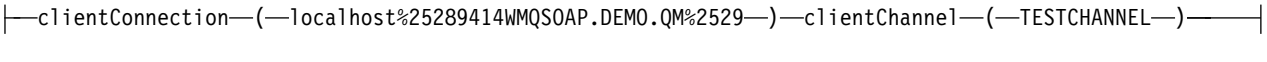
URL2.NET:

URLClient2.NET:

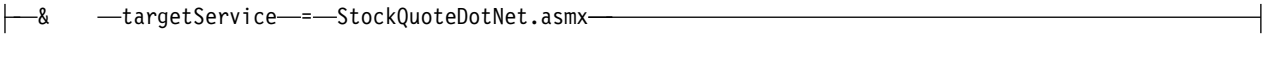
Common URL:



Client connection:



Target service:



configFile parameters

testName
The name of the test. Use *testName* in the **runivt** command

testDescription
Documentation about the test

testCommand
The command executed by the **runivt** command to make the client request.

testResponse
The exact response string returned by the client request to the console. For the test to succeed *testResponse* must match the actual response.

testListener
The name of the WebSphere MQ SOAP listener that is started by **runivt** to process the SOAP request. dotnet and JMSax are synonyms for the supplied listeners, **amqwSOAPNETlistener** and **SimpleJavaListener**.

Examples

```
runivt
```

Figure 28. run all the default tests

```
runivt dotnet
```

Figure 29. run a specific test from the default tests

```
runivt -c mytests.txt
```

Figure 30. run a set of custom tests

Related information:
Verifying the WebSphere MQ transport for SOAP

Secure Web services over WebSphere MQ transport for SOAP

You might secure Web services that use the WebSphere MQ transport for SOAP in one of two ways. Either create an SSL channel between the client and server, or use Web services security.

SSL and the WebSphere MQ transport for SOAP:

The WebSphere MQ transport for SOAP provides several SSL options that can be specified for use with client channel configured to run in SSL mode. The options differ between the .NET and Java environments. The WebSphere MQ SOAP senders and listeners process only the SSL options that are applicable to their particular environment. They ignore options that are not applicable.

The presence or absence of the `sslCipherSpec` option for .NET clients and the `sslCipherSuite` option for Java clients determines whether SSL is used or not. If the option is not specified in the URI then by default SSL is not used and all other SSL options are ignored. All SSL options are optional except where indicated.

For WebSphere MQ clients, set the SSL attributes in the URI or channel definition table. On the server, set the attributes using the facilities of WebSphere MQ.

By default, the standard WebSphere MQ SSL option, `SSLCAUTH`, is set when enabling SSL on the channel. Clients must authenticate themselves before SSL communication can commence. If `SSLCAUTH` is not set, SSL communications are established without client authentication.

To authenticate themselves, clients must have a certificate assigned in their key repository that is acceptable to the queue manager. For additional security, WebSphere MQ channels can be configured to

accept only certificates from a restricted list. The list is restricted by checking the distinguished name of the certificate against the peer name attribute of the channel.

If you use Java, the first SSL connection from a WebSphere MQ SOAP client causes the following SSL parameters to be fixed. The same values are used in subsequent connections using the same client process:

- `sslKeyStore`
- `sslKeyStorePassword`
- `sslTrustStore`
- `sslTrustStorePassword`
- `sslFipsRequired`
- `sslLDAPCRLservers`

The effect of varying these parameters on subsequent connections from this client is undefined.

If you use .NET, the first SSL connection from a WebSphere MQ SOAP client causes the following SSL parameters to be fixed. The same values are used in subsequent connections using the same client process:

- `sslKeyRepository`
- `sslCryptoHardware`
- `sslFipsRequired`
- `sslLDAPCRLservers`

The effect of varying these parameters on subsequent connections from this client is undefined. These parameters are reset if all SSL connections become inactive and a new SSL connection is made.

The following properties can also be specified as system properties:

- `sslKeyStore`
- `sslKeyStorePassword`
- `sslTrustStore`
- `sslTrustStorePassword`

If they are specified both as system properties and in the URI, and the values differ, the deployment utility displays a warning. The URI values take precedence.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

SSL connection factory parameters in the WebSphere MQ Web services URI:

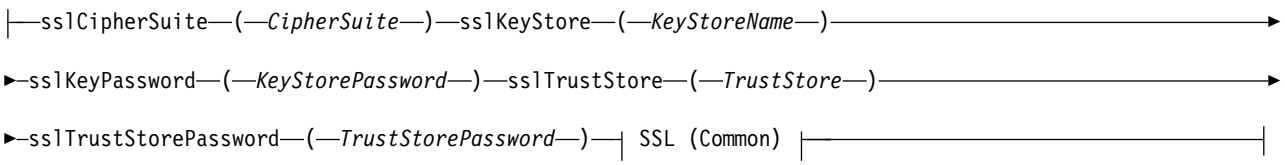
Add SSL options to the list of connection factory options in the WebSphere MQ Web services URI.

Purpose

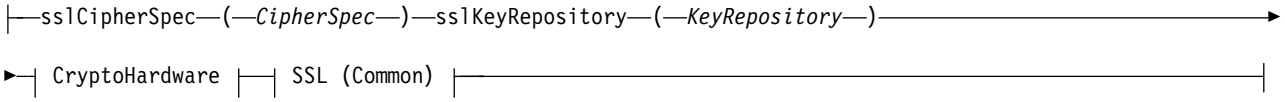
You can use a secure connection between a WebSphere MQ Web services client and the queue manager hosting the web service. The SSL options control how SSL is configured on the WebSphere MQ MQI client-server channel connection.

Syntax diagram

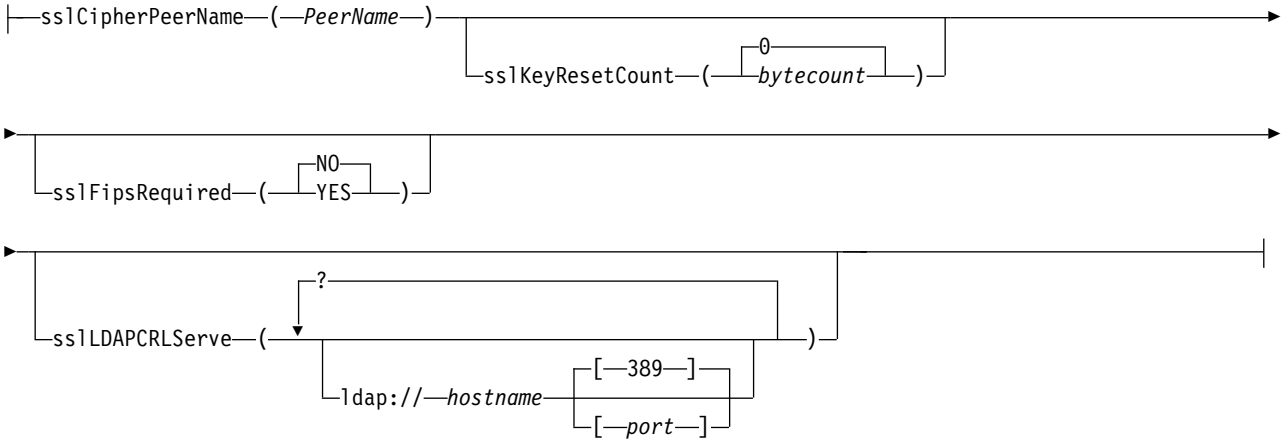
SSL (Java):



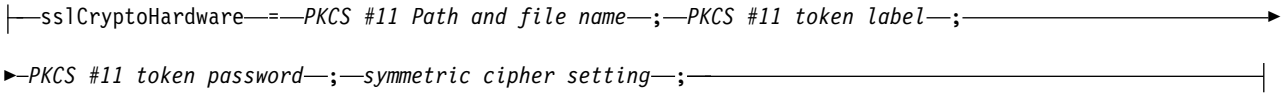
SSL (.NET):



SSL (Common):



CryptoHardware:



Required SSL parameters (Common)

sslPeerName(*peerName*)
peerName specifies the sslPeerName used on the channel.

Required SSL parameters (Java)

sslCipherSuite(*CipherSuite*)
CipherSuite specifies the sslCipherSuite used on the channel. The CipherSuite specified by the client must match the CipherSuite specified on the server connection channel.

sslKeyStore(*KeyStoreName*)
KeyStoreName specifies the sslKeyStoreName used on the channel. The keystore holds the private key

of the client used to authenticate the client to the server. The keystore is optional if the SSL connection is configured to accept anonymous client connections.

sslKeyStorePassword(*KeyStorePassword*)
KeyStorePassword specifies the sslKeyStorePassword used on the channel.

sslTrustStore(*TrustStoreName*)
TrustStoreName specifies the sslTrustStoreName used on the channel. The trust store holds the public certificate of the server, or its key chain, to authenticate the server to the client. The truststore is optional if the root certificate of a certificate authority is used to authenticate the server. In Java, root certificates are held in the JRE certificate store, `cacerts`.

sslTrustStorePassword(*TrustStorePassword*)
TrustStorePassword specifies the sslTrustStorePassword used on the channel.

Required SSL parameters (.NET)

sslCipherSpec(*CipherSpec*)
CipherSpec specifies the sslCipherSpec used on the channel. If the option is specified then SSL is used on the client channel.

sslKeyRepository(*KeyRepository*)
KeyRepository specifies the sslCipherSpec used on the channel where SSL keys and certificates are stored. *KeyRepository* is specified in stem format, that is, a full path with file name but with the file extension omitted. The effect of setting sslKeyRepository is the same as setting the `KeyRepository` field in the **MQSCO** structure on an MQCONNX call.

Optional SSL parameters (.NET)

sslCryptoHardware(*CryptoHardware*)
CryptoHardware specifies the sslCryptoHardware used on the channel. The possible values for this field, and the effect of setting it, are the same as for the `CryptoHardware` field of the **MQSCO** structure on an MQCONNX.

Optional SSL parameters (Common)

sslKeyResetCount(*bytecount*)
bytecount specifies the number of bytes passed across an SSL channel before the SSL secret key must be renegotiated. To disable the renegotiation of SSL keys omit the field or set it to zero. Zero is the only value supported in some environments, see Renegotiating the secret key in WebSphere MQ classes for Java . The effect of setting sslKeyResetCount is the same as setting the `KeyResetCount` field in the **MQSCO** structure on an MQCONNX call.

sslFipsRequired(*fipsCertified*)
fipsCertified specifies whether *CipherSpec* or *CipherSuite* must use FIPS-certified cryptography in WebSphere MQ on the channel. The effect of setting *fipsCertified* is the same as setting the `FipsRequired` field of the **MQSCO** structure on an MQCONNX call.

sslLDAPCRLServers(*LDAPServerList*)

LDAPServerList specifies a list of LDAP servers to be used for Certificate Revocation List checking.

For SSL enabled client connections, *LDAPServerList* is a list of LDAP servers to be used for Certificate Revocation List (CRL) checking. The certificate provided by the queue manager is checked against one of the listed LDAP CRL servers; if found, the connection fails. Each LDAP server is tried in turn until connectivity is established to one of them. If it is impossible to connect to any of the servers, the certificate is rejected. Once a connection has been successfully established to one of them, the certificate is accepted or rejected depending on the CRLs present on that LDAP server.

If *LDAPServerList* is blank, the certificate belonging to the queue manager is not checked against a Certificate Revocation List. An error message is displayed if the supplied list of LDAP URIs is not valid. The effect of setting this field is the same as that of including `MQAIR` records and accessing them from an **MQSCO** structure on an MQCONNX.

Related information:

Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client

Federal Information Processing Standards (FIPS) for UNIX, Linux and Windows

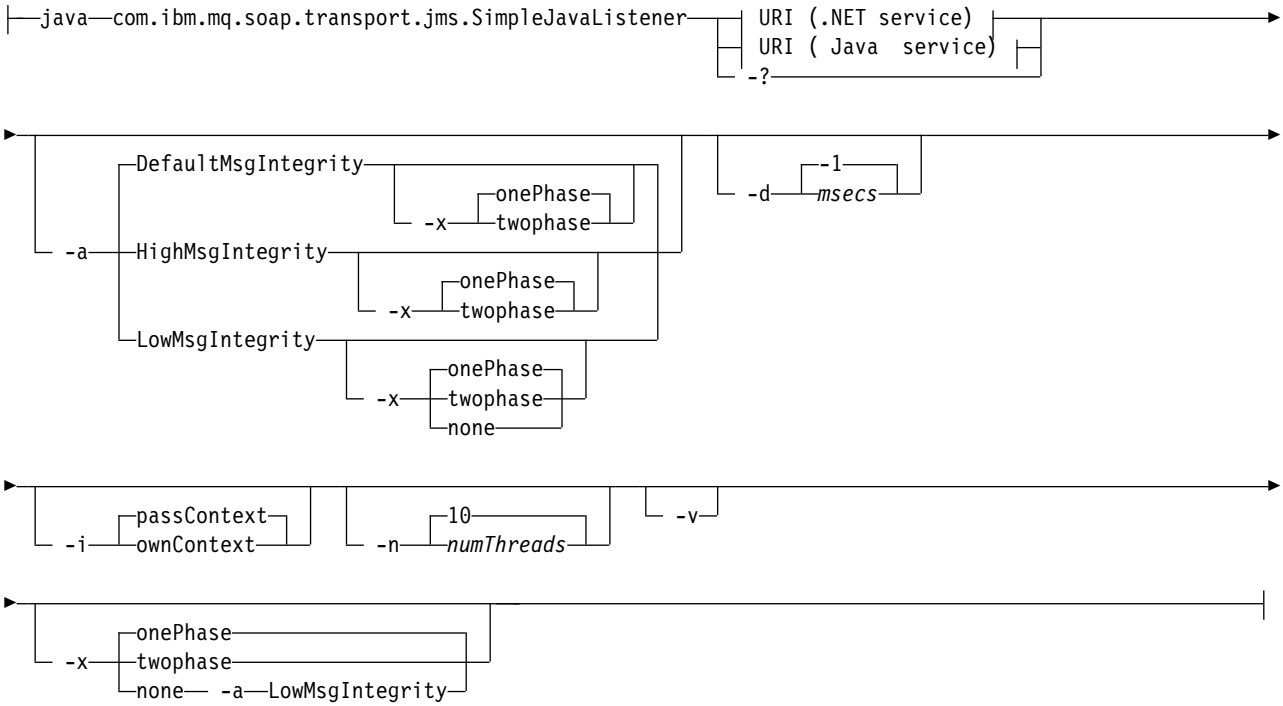
`SimpleJavaListener`: WebSphere MQ SOAP Listener for Axis 1.4

Syntax and parameters for the WebSphere MQ SOAP listener for Axis 1.4.

Purpose

Starts the WebSphere MQ SOAP listener for Axis 1.4.

Java:



Required parameters

URI *platform*
See "URI syntax and parameters for Web service deployment" on page 2307.

-? Print out help text describing how the command is used.

Optional parameters

-a *integrityOption*
integrityOption specifies the behavior of WebSphere MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

DefaultMsgIntegrity
For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an

error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

LowMsgIntegrity
For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

HighMsgIntegrity
For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

-d *msecs*
msecs specifies the number of milliseconds for the WebSphere MQ SOAP listener to stay alive if request messages have been received on any thread. If *msecs* is set to -1, the listener stays alive indefinitely.

-i *Context*
Context specifies whether the listeners pass identity context. *Context* takes the following values:

passContext
Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

ownContext
The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

-n *numThreads*
numThreads specifies the number of threads in the generated startup scripts for the WebSphere MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

-v -v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

-w *serviceDirectory*
serviceDirectory is the directory containing the web service.

-x *transactionality*
transactionality specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

onePhase
WebSphere MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. WebSphere MQ transactions ensure that the response messages are written exactly once.

twoPhase
Two-phase support is used. If the service is written appropriately the message is delivered

exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

none No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

Java Example

```
java com.ibm.mq.soap.transport.jms.SimpleJavaListener
-u "jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi"
-n 20
```

WebSphere MQ SOAP listeners

A WebSphere MQ SOAP listener reads an incoming SOAP request from the queue specified as the destination in the URI. It checks the format of the request message and then invokes a Web service using the Web services infrastructure. A WebSphere MQ SOAP listener returns any response or error from a Web service using the reply destination queue in the URI. It returns WebSphere MQ reports to the reply queue.

The term listener is used here in its standard Web services sense. It is distinct from the standard WebSphere MQ listener invoked by the **runmqlsr** command.

Description

The Java SOAP listener is implemented as a Java class and run services using Axis 1.4. The .NET listener is a console application and runs .NET Framework 1 or .NET Framework 2 services. For .NET Framework 3 services, use the WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF).

The deployment utility creates scripts to start Java or .NET SOAP listeners automatically. A SOAP Listener can be started manually using either the **amqSOAPNETListener** command, or by calling the SimpleJavaListener class. You can configure the WebSphere MQ SOAP listener to be started as a WebSphere MQ service by setting the -s option in the deployment utility. Alternatively, start listeners using triggering, or use the start and end listener scripts generated by the deployment utility. You can configure triggering manually, or use the -tmq and -tmp deployment options to configure triggering automatically. You can end a listener by setting the request queue to `GET(DISABLED)`.

Table 232. Command scripts generated by the deployment utility

Web service Infrastructure	UNIX and Linux systems	Windows Java	Windows .NET
Start listener	**startWMQJListener.sh**	**startWMQJListener.cmd**	**startWMQNListener.cmd**
Stop listener	**endWMQJListener.sh**	**endWMQJListener.cmd**	**endWMQNListener.cmd**
Define listener service	**defineWMQJListener.sh**	**defineWMQJListener.cmd**	**defineWMQNListener.cmd**

The WebSphere MQ SOAP listener passes the `endpointURL` and `soapAction` fields from the SOAP message to the SOAP infrastructure. The listener invokes the service through the Web Services infrastructure and waits for the response. The listener does not validate `endpointURL` and `soapAction`. The fields are set by the WebSphere MQ SOAP sender from the data that is provided in the URI set by a SOAP client.

The listener creates the response message and sends it to the reply destination supplied in the request message URI. In addition, the listener sets the correlation ID in the response message according to the

report option in the request message. It returns the expiry, persistence, and priority settings from the request message. The listener also sends report messages back to clients in some circumstances.

If there are format errors in the SOAP request, the listener returns a report message to the client using the reply destination queue. The queue manager also returns report messages to the client using the reply destination queue, if a report has been requested. Full report messages are written to the response queue, in response to a number of events:

- An exception.
- Message expiry.
- Format of the request message is not recognized.
- Failure of the integrity check of the **MQRFH2** header.
- The format of the main message body is not MQFMT_NONE.
- The backout/retry threshold is exceeded while the WebSphere MQ SOAP listener is processing the request.

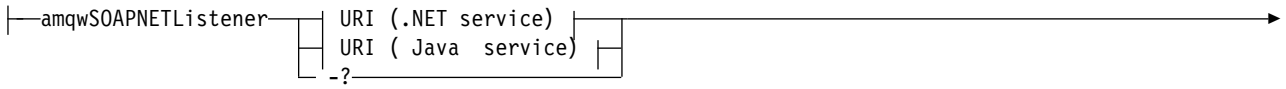
The WebSphere MQ SOAP sender sets MQRO_EXCEPTION_WITH_FULL_DATA and MQRO_EXPIRATION_WITH_FULL_DATA report options. As a result of the report options set by the WebSphere MQ SOAP sender, the report message contains the entire originating request message. The WebSphere MQ SOAP sender also sets the MQRO_DISCARD option, which causes the message to be discarded after a report message has been returned. If the report options do not meet your requirements, write your own senders to use different MQRO_EXCEPTION and MQRO_DISCARD report options. If the SOAP request is sent by a different sender that did not set MQRO_DISCARD, the failing message is written to the dead letter queue (DLQ).

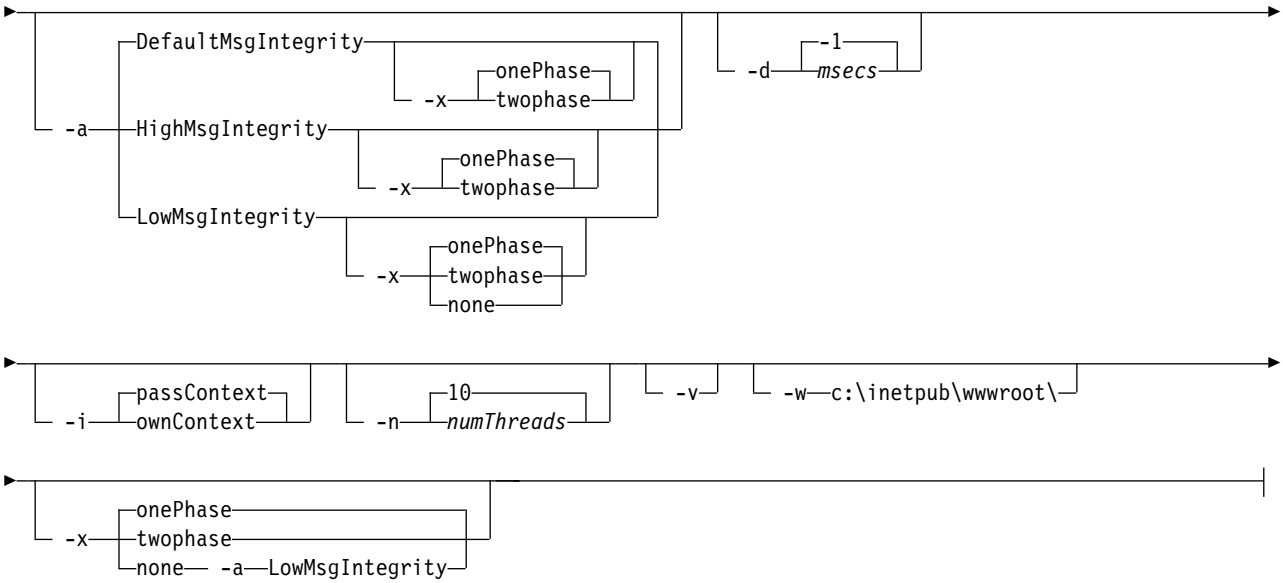
If the listener generates a report message but fails in the process of sending the report, the report message is sent to the DLQ. Ensure that your DLQ handler handles these messages correctly.

If an error occurs when attempting to write to the dead-letter queue a message is written to the WebSphere MQ error log. Whether the listener continues to process more messages depends on which message persistence and transactional options are selected. If the listener is running in one-phase transactional mode and is processing a non-persistent request message, the original message is discarded. The WebSphere MQ SOAP listener continues to execute. If the request message is persistent the request message is backed out to the request queue and the listener exits. The request queue is set to get-inhibited to prevent an accidental triggered restart.

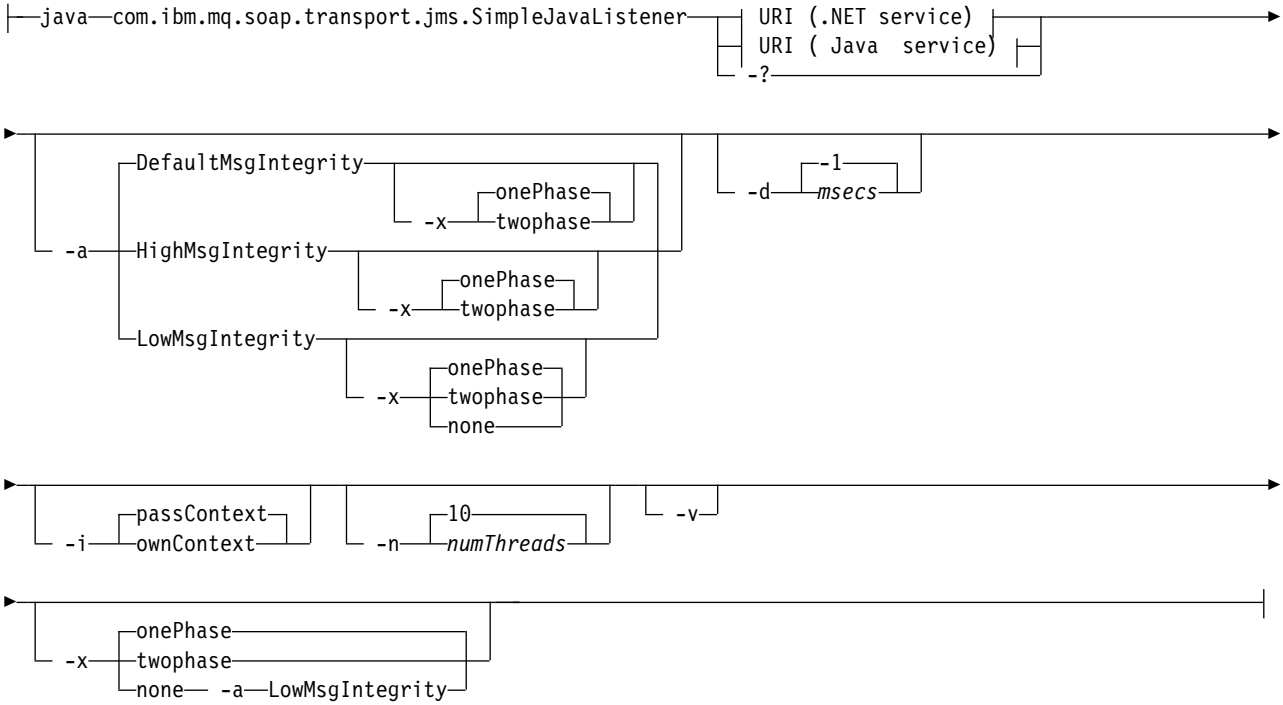
Syntax diagram

.NET:





Java:



Required parameters

URI *platform*
See "URI syntax and parameters for Web service deployment" on page 2307.

-? Print out help text describing how the command is used.

Optional parameters

-a *integrityOption*

integrityOption specifies the behavior of WebSphere MQ SOAP listeners if it is not possible to put a failed request message on the dead-letter queue. *integrityOption* can take one of the following values:

DefaultMsgIntegrity
For non-persistent messages, the listener displays a warning message and continues to execute with the original message being discarded. For persistent messages, it displays an error message, backs out the request message so it remains on the request queue and exits. DefaultMsgIntegrity applies if the -a option is omitted, or if *integrityOption* is not specified.

LowMsgIntegrity
For both persistent and non-persistent messages, the listener displays a warning and continues to execute, discarding the message.

HighMsgIntegrity
For both persistent and non-persistent messages, the listener displays an error message, backs out the request message so it remains on the request queue and exits.

The deployment utility checks for the compatibility of the -x and -a flags. If -x none is specified, then -a LowMsgIntegrity must be specified. If the flags are incompatible the deployment utility exits with an error message and with no deployment steps having been undertaken.

-d *msecs*

msecs specifies the number of milliseconds for the WebSphere MQ SOAP listener to stay alive if request messages have been received on any thread. If *msecs* is set to -1, the listener stays alive indefinitely.

-i *Context*

Context specifies whether the listeners pass identity context. *Context* takes the following values:

passContext
Set the identity context of the original request message into the response message. The SOAP listener checks that it has authority to save the context from the request queue and to pass it to the response queue. It makes the checks at run time when opening the request queue to save context, and the response queue to pass context. If it does not have the required authority, or the MQOPEN call fails, and the response message is not processed. The response message is put on the dead-letter queue with the dead-letter header containing the return code from the failed MQOPEN. The listener then continues to process subsequent incoming messages as normal.

ownContext
The SOAP listener does not pass context. The returned context reflects the user ID under which the listener is running rather than the user ID which created the original request message.

The fields in the origin context are set by the queue manager, and not by the SOAP listener.

-n *numThreads*

numThreads specifies the number of threads in the generated startup scripts for the WebSphere MQ SOAP listener. The default is 10. Consider increasing this number if you have high message throughput.

-v -v sets verbose output from external commands. Error messages are always displayed. Use -v to output commands that you can tailor to create customized deployment scripts.

-w *serviceDirectory*

serviceDirectory is the directory containing the web service.

-x *transactionality*

transactionality specifies the type of transactional control for the listener. *transactionality* can be set to one of the following values:

onePhase
WebSphere MQ one-phase support is used. If the system fails during processing, the request message is redelivered to the application. WebSphere MQ transactions ensure that the response messages are written exactly once.

twoPhase
Two-phase support is used. If the service is written appropriately the message is delivered exactly once, coordinated with other resources, within a single committed execution of the service. This option applies to server bindings connections only.

none No transactional support. If the system fails during processing, the request message can be lost, even if persistent. The service might or might not have executed, and response, report or dead-letter messages might or might not be written.

The deployment utility checks for the compatibility of the -x and -a flags. See the description of the -a flag for details.

.NET Example

```
amqwSOAPNETlistener
-u "jms:/queue?destination=myQ&connectionFactory=()
&targetService=myService&initialContextFactory=com.ibm.mq.jms.Nojndi"
-w C:/wmqsoap/demos
-n 20
```

Java Example

```
java com.ibm.mq.soap.transport.jms.SimpleJavaListener
-u "jms:/queue?destination=myQ&connectionFactory=()
&initialContextFactory=com.ibm.mq.jms.Nojndi"
-n 20
```

WebSphere MQ transport for SOAP sender

Sender classes are provided for Axis and .NET Framework 1 and .NET Framework 2. The sender constructs a SOAP request and puts it on a queue, it then blocks until it has read a response from the response queue. You can alter the behavior of the classes by passing different URIs from a SOAP client. For .NET Framework 3 use WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF).

Purpose

The WebSphere MQ SOAP sender puts a SOAP request to invoke a Web service onto a WebSphere MQ request queue. The sender sets fields in the **MQRFH2** header according to options specified in the URI, or according to defaults.

If you need to change the behavior of a sender beyond what is possible using the URI options, write your own sender. Your sender can work with the WebSphere MQ transport for SOAP listeners, or with other SOAP environments. Your sender must construct SOAP messages in the format defined by WebSphere MQ. The format is supported by the WebSphere MQ SOAP listener, and also SOAP listeners provided by WebSphere Application Server and CICS. Your sender must follow the rules for an WebSphere MQ requestor. The WebSphere MQ SOAP listener returns reply and report messages. See "**MQMD** SOAP settings" on page 2286 for details how to set the report options in the **MQMD**. The report options control the report messages returned by the WebSphere MQ SOAP listener.

Description

The WebSphere MQ SOAP Java sender is registered with the Axis host environment for the `jms:` URI prefix. The sender is implemented in the class `com.ibm.mq.soap.transport.jms.WMQSender`, which is derived from `org.apache.axis.handlers.BasicHandler`. If the Axis host environment detects a `jms:` URI prefix it invokes the `com.ibm.mq.soap.transport.jms.WMQSender` class. The class blocks after placing the message until it has read a response from the response queue. If no response is received within a timeout interval the sender throws an exception. If a response is received within the timeout interval the response message is returned to the client using the Axis framework. Your client application must be able to handle these response messages.

For Microsoft .NET Framework 1 and .NET Framework 2 services, the WebSphere MQ SOAP sender is implemented in the class `IBM.WMQSOAP.MQWebRequest`, which is derived from `System.Net.WebRequest` and `System.Net.IwebRequestCreate`. If .NET Framework 1 or .NET Framework 2 detects a `jms:` URI prefix it invokes the `IBM.WMQSOAP.MQWebRequest` class. The sender creates an `MQWebResponse` object to read the response message from the response queue and return it to the client.

`com.ibm.mq.soap.transport.jms.WMQSender` is a final class, and `IBM.WMQSOAP.MQWebRequest` is sealed. You cannot modify their behavior by creating subclasses.

Parameters

Set the URI to control the behavior of the WebSphere MQ SOAP sender in a Web service SOAP client. The deployment utility creates Web service client stubs incorporating the URI options supplied to the deployment utility.

Use a channel definition table with the WebSphere MQ SOAP transport for SOAP sender:

A client connection channel definition is an alternative to setting connection properties in the ConnectionFactory attribute of the Web service URI. The connection properties are clientChannel, clientConnection, and SSL parameters.

Description

Create the client channel description table by defining client connections. Even if a Web services client connects to different queue managers, create all the connections in the connection table on a single queue manager. The default name and location of the connection table is *queue manager directory*`/@ipcc/AMQCLCHL.TAB`.

Pass the location of the connection table to a Java client by setting the `com.ibm.mq.soap.transport.jms.mqchlurl` system property.

Pass the location of the connection table to a .NET client by setting the `MQCHLLIB` and `MQCHLTAB` environment variables.

You might provide both a channel connection table and channel connection parameters in the ConnectionFactory attribute of the Web service URI. The values set in the ConnectionFactory take precedence over the values in the channel definition table.

Using a channel definition table in Java

```
java -Dcom.ibm.msg.client.config.location=file:/C:/mydir/myjms.config MyAppClass
```

Figure 31. Starting Java client using a configuration file

```
com.ibm.mq.soap.transport.jms.mqchlurl=file:/C:/ibm/wmq/qmgrs/QM1/@ipcc/AMQCLCHL.TAB
```

Figure 32. myjms.config

Transactions

Use the `-x` option when starting the listener, to run Web services transactionally. Select the integrity of messages by setting the `persistence` option in the service URI.

Web services

Use the `-x` option when starting the listener, to run Web services transactionally. On .NET Framework 1 and 2 the SOAP listener uses Microsoft Transaction Coordinator (MTS). On Axis 1.4, the SOAP listener uses queue manager coordinated transactions.

Web service clients

The SOAP senders are not transactional.

WebSphere MQ bindings

You can set the binding type for the SOAP sender. It can connect as a WebSphere MQ server application, or as a client application. You can also bind the SOAP sender as an XA-client on .NET.

Message persistence

Select the level of persistence by setting the `Persistence` option in the URI.

Web service transactions

You can use Web service transactions, because the SOAP sender is not transactional. If you write your own SOAP sender, and intend to use Web service transactions, do not create a transactional SOAP sender. You cannot send the request message and receive the reply message in the same transaction. The send and receive must not be coordinated by the Web service transaction.

URI syntax and parameters for Web service deployment

The syntax and parameters to deploy a IBM WebSphere MQ Web service are defined in a URI. The deployment utility generates a default URI based on the name of the Web service. You can override the defaults by defining your own URI as a parameter to the deployment utility. The deployment utility incorporates the URI in the generated Web service client stubs.

Purpose

A web service is specified using a Universal Resource Identifier (URI). The syntax diagram specifies the URI that is supported in the IBM WebSphere MQ transport for SOAP. The URI controls over IBM WebSphere MQ-specific SOAP parameters and options used to access target services. The URI is compatible with Web services hosted by .NET, Apache Axis 1, WebSphere Application Server, CICS.

Description

The URI is incorporated into the Web service client classes generated by the deployment utility. The client passes the URI to IBM WebSphere MQ SOAP Sender in a IBM WebSphere MQ message. The URI controls

the processing performed by both the IBM WebSphere MQ SOAP Sender and IBM WebSphere MQ SOAP listener.

Syntax

The URI syntax is as follows:

jms:/queue?name=*value***&name=***value***...**

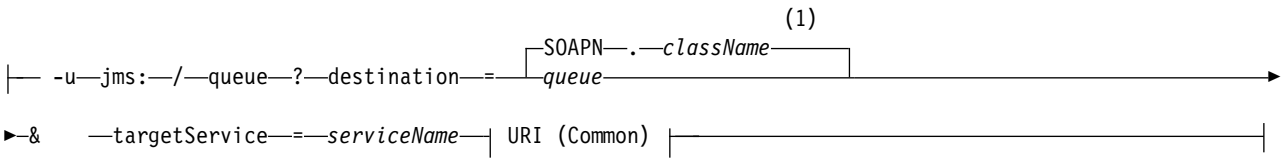
where **name** is a parameter name and *value* is an appropriate value, and the **name**=*value* element can be repeated any number of times with the second and subsequent occurrences being preceded by an ampersand (&).

Parameter names are case-sensitive, as are names of IBM WebSphere MQ objects. If any parameter is specified more than once, the final occurrence of the parameter takes effect. Client applications can override a generated parameter by appending another copy of the parameter to the URI. If any additional unrecognized parameters are included, they are ignored.

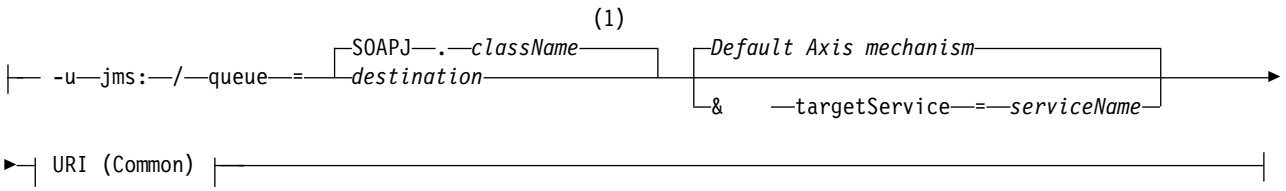
If you store a URI in an XML string, you must represent the ampersand character as `&`. Similarly, if a URI is coded in a script, take care to escape characters such as & that would otherwise be interpreted by the shell.

Syntax diagram

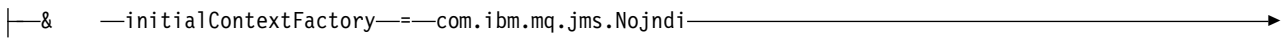
URI (.NET service):

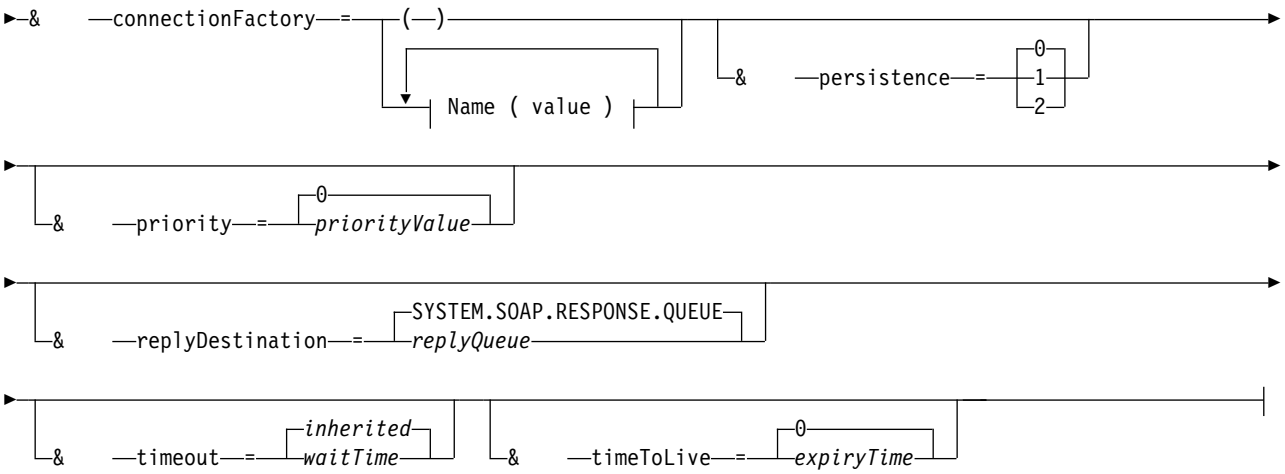


URI (Java service):

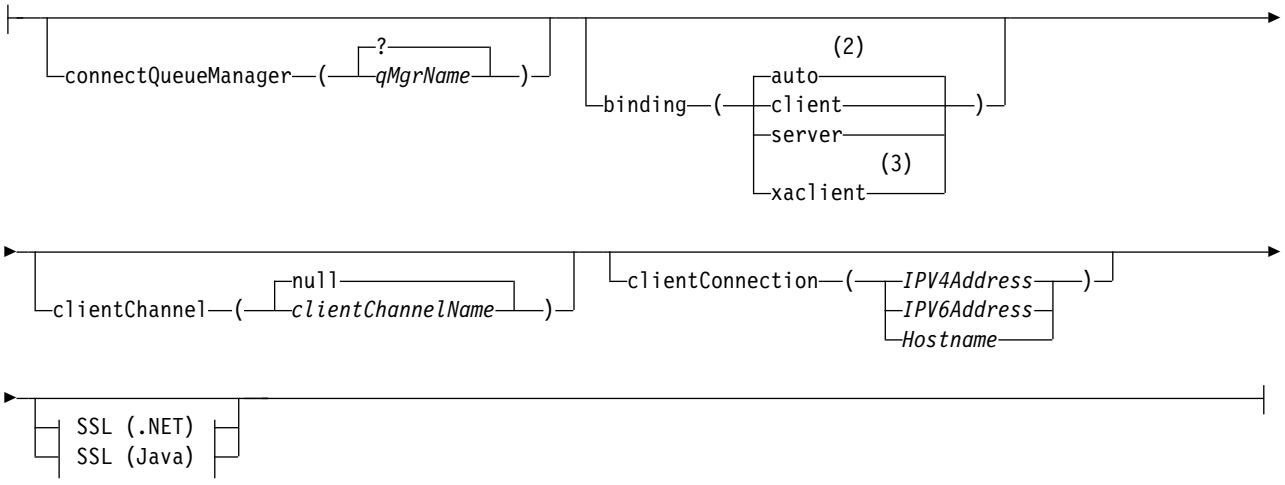


URI (Common):





Name (value):



Notes:

1 The queue manager transforms *className* to a queue name following the steps described in "Destination to queue name transformation"

2 **client** is the default if other options appropriate for a client are specified; for example **clientConnection**.

3 **xaclient** applies to .NET only

Destination to queue name transformation

1. *className* is prefixed with `SOAPJ.` for Java services or with `SOAPN.` for .NET services.
2. The file extension is removed from the full path name given in the *className* parameter.
3. The resulting string is truncated to no more than 48 characters
4. Directory separator characters are replaced with period characters.
5. Embedded spaces are replaced with underscore characters.
6. The colon following a drive prefix letter is replaced with a period for a .NET service.

Note: In some environments, a queue name generated by the deployment utility might not be unique. The deployment utility makes checks whether to create the queue. You might choose to override the deployment utility by restructuring the deployment directory hierarchy, or by customizing the supplied deployment process.

Required URI parameters

destination=*queue*

queue is the name of the request destination. It can be a queue or a queue alias. If it is queue alias, the alias can resolve to a topic.

- If the -u parameter is omitted *queue* is generated from *classname* using the steps described in "Destination to queue name transformation" on page 2309.
- If the-u parameter is specified *queue* is required and must be the first parameter of the URI after the initial **jms:/queue**? string. Specify either a IBM WebSphere MQ queue name, or a queue name and queue manager name connected by an @ symbol, for example `SOAPN.trandemos@WMQSOAP.DEMO.QM`.
- The deployment utility checks whether the queue name, generated or provided, matches the name of an existing queue. The action taken is described in Table 233.

Table 233. queue validation

Listener script exists?	**Listener script exists in the `./generated/server` directory**		**Listener script does not exist in the `./generated/server` directory**
Queue in listener script matches *queue*?	***queue* does not match request queue being used in listener script**	***queue* matches request queue being used in listener script**	
queue exists	• *Deployment exits with an error.* • The service has already been deployed in `./generated/server`, but using a different queue.	• **Deployment continues normally.** • The service has already been deployed in `./generated/server`	• *Deployment exits with an error.* • The listener startup script is not found in `./generated/server`, but *queue* is in use by a different service or application.
queue does not exist		• **Deployment continues with a warning.** • The previous deployment might have failed, because the startup is valid, but *queue* is missing.	• **Deployment continues normally.** • No service has been deployed from this directory.

&connectionFactory=*Name(value)*

Name is one of the following parameters:

- connectQueueManager(qMgrName)
- binding(bindingType)
- clientChannel(channel)
- clientConnection(connection)
- "Required SSL parameters (Java)" on page 2297

See "Connection factory parameters" on page 2312 for a description of the values of these parameters.

&targetService=*serviceName*

[7]On .NET, *serviceName* is the name of a .NET service located in the deployment directory, for example: `targetService=myService.asmx`. In the .NET environment, the targetService parameter makes it possible for a single WebSphere MQ SOAP listener to be able to process requests for multiple services. These services must be deployed from the same directory.

7. .NET service only

Optional URI parameters

&initialContextFactory=*contextFactory*

contextFactory is required and must be set to `com.ibm.mq.jms.Nojndi`. Make sure `Nojndi.jar` is in the class path for a WebSphere Application Server Web services client. `Nojndi.jar` returns Java objects based on the contents of the **connectionFactory** and **destination** parameters, rather than by reference to a directory.

&targetService=*serviceName*

[8]On Axis, *serviceName* is the fully qualified name of a Java service, for example: `targetService=javaDemos.service.StockQuoteAxis`. If targetService is not specified, a service is loaded using the default Axis mechanism.

&persistence=*messagePersistence*

messagePersistence takes one of the following values:

0 Persistence is inherited from the queue definition.

1 The message is non-persistent.

2 The message is persistent

&priority=*priorityValue*

priorityValue is in the range 0 - 9. 0 is low priority. The default value is environment-specific, which in the case of IBM WebSphere MQ is 0.

&replyDestination=*replyToQueue*

The queue at the client side to be used for the response message. The default reply queue is `SYSTEM.SOAP.RESPONSE.QUEUE`.

- Run the `setupWMQSOAP` script to create the default WebSphere MQ SOAP objects.
- Specify a model queue for *replyToQueue* to create either a temporary or permanent dynamic response queue. For both temporary and permanent dynamic response queues, a separate instance of dynamic queue is created for each request. If any of the following events happen the queue is deleted:
  - The response arrives and is processed.
  - The request times out.
  - The requesting program terminates.

  For the best performance, use temporary dynamic queues rather than permanent dynamic queues. Do not send a persistent request message to a URI with a temporary dynamic queue. The IBM WebSphere MQ listener SOAP fails to process the message and outputs an error. The client times out waiting for the reply.
- The `setupWMQSOAP` script creates a default permanent dynamic model queue called `SYSTEM.SOAP.MODEL.RESPONSE.QUEUE`.

&timeout=*waitTime*

The time, in milliseconds, that the client waits for a response message. *waitTime* overrides values set by the infrastructure or client application. If not specified the application value, if specified, or infrastructure default is inherited.

Note: No relationship is enforced between timeout and timeToLive.

&timeToLive=*expiryTime*

expiryTime is the time, specified in milliseconds, before the message expires. The default is zero, which indicates an unlimited lifetime.

Note: No relationship is enforced between timeout and timeToLive.

8. Java service only

Connection factory parameters

connectQueueManager(*qMgrName*)
qMgrName specifies the queue manager to which the client connects. The default is blank.

binding(*bindingType*)
bindingType specifies how the client is connected to *qMgrName*. The default is auto. *bindingType* takes the following values:

auto The sender tries the following connection types, in order:

1. If other options appropriate to a client connection are specified, the sender uses a client binding. The other options are clientConnection or clientChannel.
2. Use a server connection.
3. Use a client connection.

Use **binding**(*auto*) in the *URI* if there is no local queue manager at the SOAP client. A client connection is built for the SOAP client.

client Use **binding**(*client*) in the *URI* to build a client configuration for the SOAP sender.

server Use **binding**(*server*) in the *URI* to build a server configuration for the SOAP sender. If the connection has client type parameters the connection fails and an error is displayed by the IBM WebSphere MQ SOAP sender. Client type parameters are clientConnection, clientChannel, or SSL parameters.

xaclient
xaclient is applicable only on .NET and not for Java clients. Use an XA-client connection.

clientChannel(*channel*)

The SOAP client uses *channel* to make a IBM WebSphere MQ client connection. *channel* must match the name of a server connection channel, unless channel auto definition is enabled at the server. clientChannel is a required parameter, unless you have provided a Client Connection Definition table (CCDT).

Provide a CCDT in Java by setting `com.ibm.mq.soap.transport.jms.mqchlurl`. In .NET set the `MQCHLLIB` and `MQCHLTAB` environment variables; see "Use a channel definition table with the WebSphere MQ SOAP transport for SOAP sender" on page 2306.

clientConnection(*connection*)
The SOAP client uses *connection* to make a IBM WebSphere MQ client connection. The default hostname is localhost, and default port is 1414. If *connection* is a TCP/IP address, it takes one of three formats, and can be suffixed with a port number.

JMS clients can either use the format: `hostname:port` or 'escape' the brackets using the format `%X` where `X` is the hexadecimal value that represents the bracket character in the code page of the URI. For example, in ASCII, `%28` and `%29` for `(` and `)` respectively.

.Net clients can use the brackets explicitly: `hostname(port)` or use the 'escaped' format.

IPV4 address
For example, `192.0.2.0`.

IPV6 address
For example, `2001:DB8:0:0:0:0:0:0`.

Host name
For example, `www.example.com%281687%29`, `www.example.com:1687`, or `www.example.com(1687)`.

SSL *platform*
See "Required SSL parameters (Java)" on page 2297

Sample URIs

Note:

1. & in the URI is encoded as &
2. All the parameter listed previously are applicable to the clients.
3. Only **destination**, **connectionFactory** and **initialContextFactory** are applicable to the WCF service.

```
jms:/queue?destination=myQ&amp;amp;connectionFactory=()&amp;amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 33. URI for an Axis service, supplying only required parameters

```
jms:/queue?destination=myQ&amp;amp;connectionFactory=()&amp;amp;targetService=MyService.asmx
&amp;amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 34. URI for a .NET service, supplying only required parameters

```
jms:/queue?destination=myQ@myRQM&amp;amp;connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
&amp;amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 35. URI for an Axis service, supplying some optional connectionFactory parameters

```
jms:/queue?destination=myQ@myRQM&amp;amp;connectionFactory=connectQueueManager(myconnQM)
binding(client)clientChannel(myChannel)clientConnection(myConnection)
sslPeerName(CN=MQ Test 1,O=IBM,S=Hampshire,C=GB)
&amp;amp;initialContextFactory=com.ibm.mq.jms.Nojndi
```

Figure 36. URI for an Axis service, supplying the sslPeerName option of the connectionFactory parameter

The Nojndi mechanism:

The Nojndi mechanism enables JMS programs, which use JNDI interfaces, to use the same URI as WebSphere MQ programs, which do not use JNDI.

You can use the WebSphere MQ transport for SOAP to invoke Web services on WebSphere Application Server. WebSphere Application Server SOAP over JMS looks up the JMS resources using JNDI. The Web service client might be running on .NET, or using Axis 1.4, to invoke the Web service and not using JNDI. To use the same URL for the client and server, it must provide the same information whether the environment is using JNDI or not.

The URI passed to the WebSphere MQ transport for SOAP by a Web service client contains a specific WebSphere MQ queue manager and queue names. These names are parsed and used directly by WebSphere MQ SOAP support.

The Nojndi mechanism directs the initialContextFactory used by a JMS program to com.ibm.mq.jms.Nojndi. The com.ibm.mq.jms.Nojndi class is an implementation of the JNDI interface that returns the connectionFactory and destination from the URL as ConnectionFactory and Queue Java objects. If the JMS implementation is WebSphere MQ, MQConnectionFactory and MQQueue inherit from the ConnectionFactory and Queue classes.

By using the Nojndi mechanism, you are able to provide the same connection information to WebSphere Application server and .NET using the same URL.

W3C SOAP over JMS URI for the WebSphere MQ Axis 2 client

Define a W3C SOAP over JMS URI to call a Web service from an Axis 2 client using WebSphere MQ JMS as the SOAP transport. The Web service must be provided by a server that supports WebSphere MQ JMS and the W3C SOAP over JMS candidate recommendation for the SOAP/JMS binding.

Description

The W3C candidate recommendation defines the SOAP over JMS binding; SOAP over Java Message Service 1.0. Also useful for its examples is URI Scheme for Java(tm) Message Service 1.0[9].

Use the syntax diagram to create W3C SOAP over JMS URIs that are syntactically correct, and are accepted by the WebSphere MQ Axis 2 client. It is limited to defining the URI that is accepted by the WebSphere MQ Axis 2 client. It is a subset of the W3C recommendation in two respects:

1. The jms-variant topic is not supported, and must not be specified in a URI passed to the WebSphere MQ Axis 2 client.
2. The following properties are omitted from the syntax diagram because they are JMS properties, and not part of the URI.
   a. bindingVersion
   b. contentType
   c. soapAction
   d. requestURI
   e. isFault

The JMS properties are set by the Axis 2 client or the server.

The diagram extends the W3C recommendation by defining a custom parameter, connectionFactory. connectionFactory is used as an alternative to JNDI to specify how the Axis 2 client connects to a queue manager using a queue.

The WebSphere MQ Axis 2 client only accepts properties as part of the URI passed to the client by the client application or as environment variables. The WebSphere MQ Axis 2 client has no capability to process a WSDL document. The client application or a development tool might process the WSDL and create the URI to pass to the Axis 2 client. An WebSphere MQ Axis 2 client application cannot set the JMS message properties directly.

Syntax

In accordance with the W3C recommendation, all the parameters can be obtained from environment variables. The environment variable names are formed by prefacing the parameter name with `soapjms_`. The syntax is: **soapjms**_*parameterName*; for example,

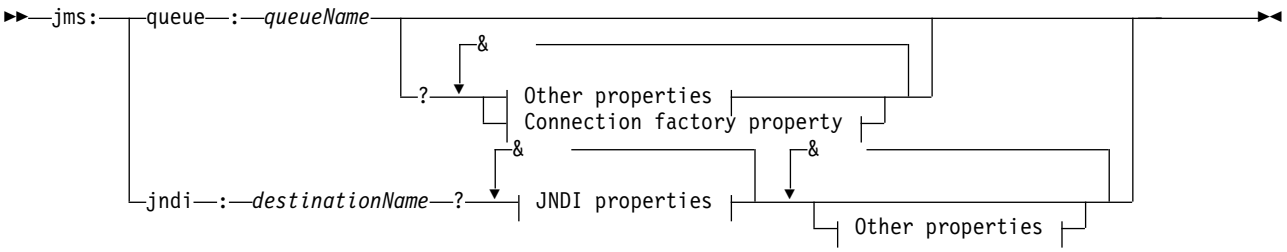
```
set soapjms_targetServer=com.example.org.stockquote
```

If a parameter is set using an environment variable it overrides the value set in the URI.

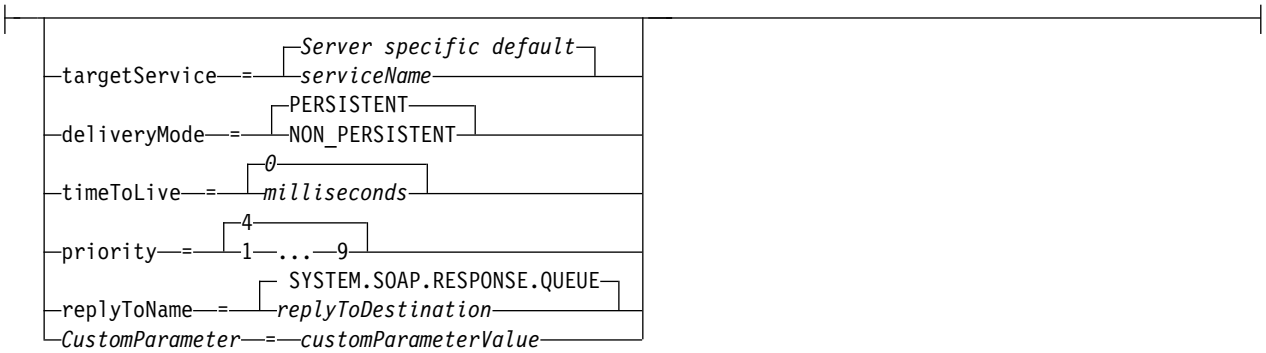
In accordance with the W3C recommendation, all the parameters can be repeated. The last instance of a parameter is used, unless overridden by an environment variable.

9. Look for *URI Scheme for JMS*, in the W3C specification references, for the latest draft.

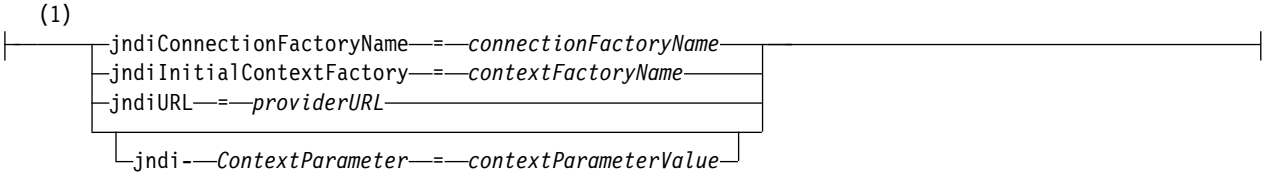
jms-uri



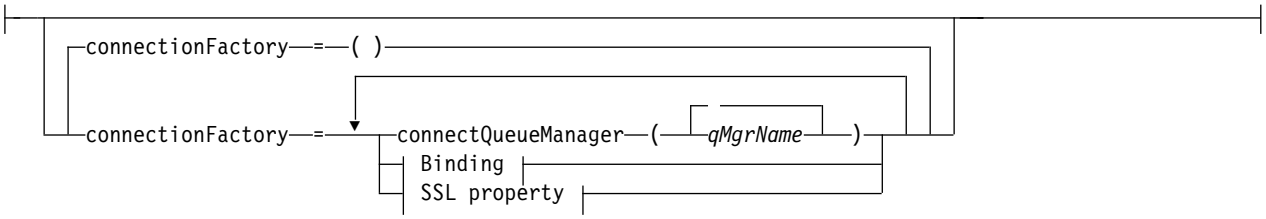
Other properties:



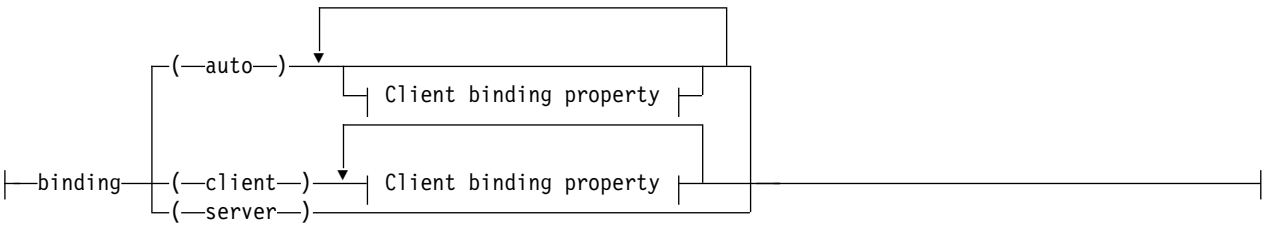
JNDI properties:



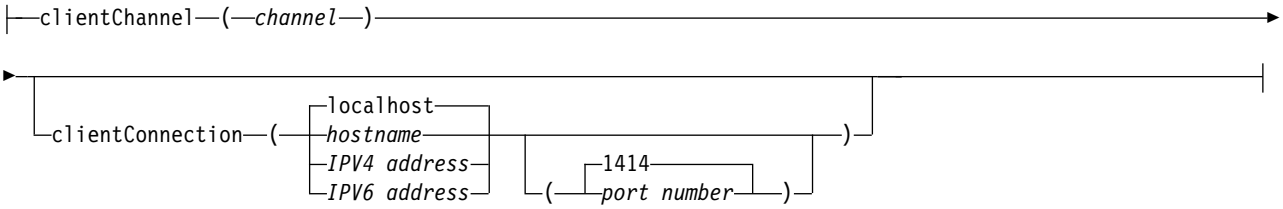
Connection factory property:



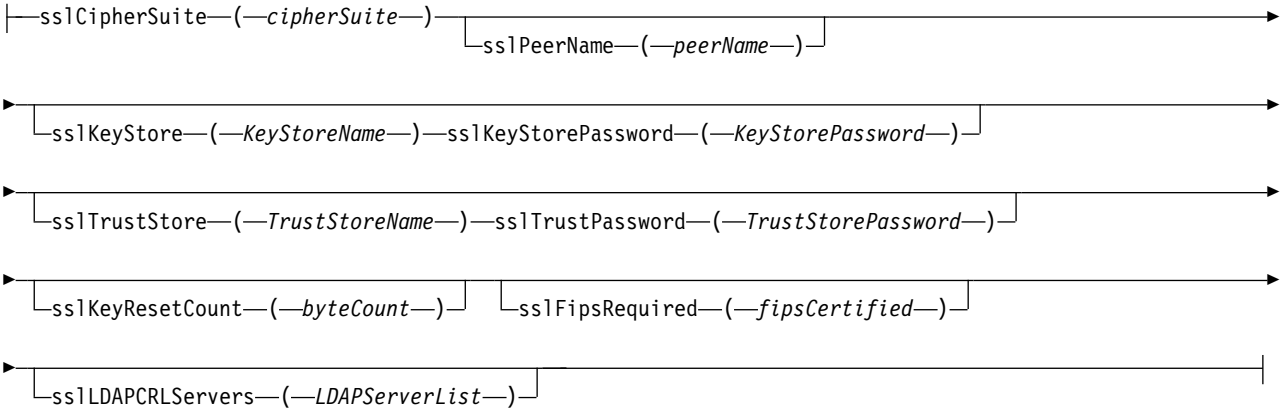
Binding:



Client binding property:



SSL property:



Notes:

1 **jndiConnectionFactoryName**, **jndiConnectionFactoryName** and **jndiURL** are all required parameters. **jndi-***ContextParameter* is optional.

Parameters

connectionFactory=*connectionFactoryParameterList*

connectionFactoryParameterList are parameters that qualify how the Axis 2 client connects to a queue manager when the destination variant is queue.

connectionFactory must not be specified with the jndi destination variant.

The parameters are not passed to the server in the request URI.

If connectionFactory is omitted, the queue must belong to a default queue manager running on the same server as the Axis 2 client.

The *connectionFactoryParameterList*:

binding(*bindingType*)

bindingType specifies how the client is connected to *qMgrName*. The default is auto. *bindingType* takes the following values:

auto The sender tries the following connection types, in order:

1. If other options appropriate to a client connection are specified, the sender uses a client binding. The other options are clientConnection or clientChannel.
2. Use a server connection.
3. Use a client connection.

Use **binding**(*auto*) in the *URI* if there is no local queue manager at the SOAP client. A client connection is built for the SOAP client.

client Use **binding**(*client*) in the *URI* to build a client configuration for the SOAP sender.

server Use **binding**(*server*) in the *URI* to build a server configuration for the SOAP sender. If the connection has client type parameters the connection fails and an error is displayed by the IBM WebSphere MQ SOAP sender. Client type parameters are clientConnection, clientChannel, or SSL parameters.

xaclient
xaclient is applicable only on .NET and not for Java clients. Use an XA-client connection.

clientChannel(*channel*)

The SOAP client uses *channel* to make a IBM WebSphere MQ client connection. *channel* must match the name of a server connection channel, unless channel auto definition is enabled at the server. clientChannel is a required parameter, unless you have provided a Client Connection Definition table (CCDT).

Provide a CCDT in Java by setting `com.ibm.mq.soap.transport.jms.mqchlurl`. In .NET set the `MQCHLLIB` and `MQCHLTAB` environment variables; see "Use a channel definition table with the WebSphere MQ SOAP transport for SOAP sender" on page 2306.

clientConnection(*connection*)
The SOAP client uses *connection* to make a IBM WebSphere MQ client connection. The default hostname is localhost, and default port is 1414. If *connection* is a TCP/IP address, it takes one of three formats, and can be suffixed with a port number.

JMS clients can either use the format: `hostname:port` or 'escape' the brackets using the format `%X` where `X` is the hexadecimal value that represents the bracket character in the code page of the URI. For example, in ASCII, `%28` and `%29` for `(` and `)` respectively.

.Net clients can use the brackets explicitly: `hostname(port)` or use the 'escaped' format.

IPV4 address
For example, `192.0.2.0`.

IPV6 address
For example, `2001:DB8:0:0:0:0:0:0`.

Host name
For example, `www.example.com%281687%29`, `www.example.com:1687`, or `www.example.com(1687)`.

sslCipherSuite(*CipherSuite*)
CipherSuite specifies the sslCipherSuite used on the channel. The CipherSuite specified by the client must match the CipherSuite specified on the server connection channel.

sslFipsRequired(*fipsCertified*)
fipsCertified specifies whether *CipherSpec* or *CipherSuite* must use FIPS-certified cryptography in WebSphere MQ on the channel. The effect of setting *fipsCertified* is the same as setting the `FipsRequired` field of the **MQSCO** structure on an MQCONNX call.

sslKeyStore(*KeyStoreName*)
KeyStoreName specifies the sslKeyStoreName used on the channel. The keystore holds the private key of the client used to authenticate the client to the server. The keystore is optional if the SSL connection is configured to accept anonymous client connections.

sslKeyResetCount(*bytecount*)
bytecount specifies the number of bytes passed across an SSL channel before the SSL secret key must be renegotiated. To disable the renegotiation of SSL keys omit the field or set it to zero. Zero is the only value supported in some environments, see Renegotiating the secret key in WebSphere MQ classes for Java . The effect of setting sslKeyResetCount is the same as setting the `KeyResetCount` field in the **MQSCO** structure on an MQCONNX call.

sslKeyStorePassword(*KeyStorePassword*)
KeyStorePassword specifies the sslKeyStorePassword used on the channel.

sslLDAPCRLServers(*LDAPServerList*)

LDAPServerList specifies a list of LDAP servers to be used for Certificate Revocation List checking.

For SSL enabled client connections, *LDAPServerList* is a list of LDAP servers to be used for Certificate Revocation List (CRL) checking. The certificate provided by the queue manager is checked against one of the listed LDAP CRL servers; if found, the connection fails. Each LDAP server is tried in turn until connectivity is established to one of them. If it is impossible to connect to any of the servers, the certificate is rejected. Once a connection has been successfully established to one of them, the certificate is accepted or rejected depending on the CRLs present on that LDAP server.

If *LDAPServerList* is blank, the certificate belonging to the queue manager is not checked against a Certificate Revocation List. An error message is displayed if the supplied list of LDAP URIs is not valid. The effect of setting this field is the same as that of including `MQAIR` records and accessing them from an **MQSCO** structure on an MQCONNX.

sslPeerName(*peerName*)
peerName specifies the sslPeerName used on the channel.

sslTrustStore(*TrustStoreName*)
TrustStoreName specifies the sslTrustStoreName used on the channel. The trust store holds the public certificate of the server, or its key chain, to authenticate the server to the client. The truststore is optional if the root certificate of a certificate authority is used to authenticate the server. In Java, root certificates are held in the JRE certificate store, `cacerts`.

sslTrustStorePassword(*TrustStorePassword*)
TrustStorePassword specifies the sslTrustStorePassword used on the channel.

CustomParameter=customParameterValue

CustomParameter is the user-defined name of a custom parameter, and *customParameterValue* is the value of the parameter.

Custom parameters that are not used by the Axis 2 client are sent by the Axis 2 client to the SOAP server. Consult the server documentation. connectionFactory is a custom parameter that is used by the Axis 2 client and is not passed to the server.

CustomParameter must not match the name of an existing parameter.

If *CustomParameter* starts with the string `jndi-` it is used in looking up a JNDI destination; see jndi-.

deliveryMode=*deliveryMode*
deliveryMode sets the message persistence. The default is `PERSISTENT`.

jndi:*destinationName*
destinationName is a JNDI destination name that maps to a JMS queue. If the jndi destination variant is specified, you must provide a *destinationName*.

jndiConnectionFactoryName=*connectionFactoryName*
connectionFactoryName sets the JNDI name of the connection factory. If the destination variant is jndi, *connectionFactoryName* must be provided.

jndiInitialContextFactory=*contextFactoryName*
contextFactoryName sets the JNDI name of the initial context factory. If the destination variant is jndi, *contextFactoryName* must be provided. See Using JNDI to retrieve administered objects in a JMS application.

jndiURL=*providerURL*
jndiURL sets the URL name of the JNDI provider. If the destination variant is jndi, *jndiURL* must be specified.

jndi-*ContextParameter=contextParameterValue*
jndi-*ContextParameter* is the user-defined name of a custom parameter that used to pass information to the JNDI provider. *contextParameterValue* is the information that is passed.

priority=*priorityValue*
priorityValue sets the JMS message priority. `0` is low, `9` is high. The default value is `4`.

queue:*queueName*
queueName is the name of a JMS queue on which the SOAP request is placed. If the queue variant is specified, a queue name must be provided. If the queue does not belong to a default queue manager on the same server as the client, set the connectionFactory parameter.

replyToName=*replyToDestination*
replyToDestination sets the destination queue name. If the destination variant is jndi, the name is a JNDI name that must map to a queue. If the variant is queue the name is a JMS queue. The default value is `SYSTEM.SOAP.RESPONSE.QUEUE`.

targetService=*serviceName*
The name used by the SOAP server to start the target Web service.

On Axis, *serviceName* is the fully qualified name of a Java service, for example: `targetService=www.example.org.StockQuote`. If targetService is not specified, a service is loaded using the default Axis mechanism.

timeToLive=*milliseconds*
Set *milliseconds* to the time before the message expires. The default, `0`, is the message never expires.

Examples

```
jms:jndi:REQUESTQ
    ?jndiURL=file:/C:/JMSAdmin
    &jndiInitialContextFactory=com.sun.jndi.fscontext.RefFSContextFactory
    &jndiConnectionFactoryName=ConnectionFactory
    &replyToName=RESPONSEQ
    &deliveryMode(NON_PERSISTENT)
```

Figure 37. Use jms:jndi to send a SOAP/JMS request

```
jms:queue:SOAPJ.demos
    ?connectionFactory=connectQueueManager(QM1)
     Bind(Client)
     ClientChannel(SOAPClient)
     ClientConnection(www.example.org(1418))
    &deliveryMode(NON_PERSISTENT)
```

Figure 38. Use jms:queue to send a SOAP/JMS request

Supported Web services

Code that has been written to run as a Web service does not need to be modified to use the WebSphere MQ transport for SOAP. You do need to deploy services differently to run with the WebSphere MQ transport for SOAP rather than using HTTP.

Description

The WebSphere MQ transport for SOAP provides a SOAP listener to run services for .NET Framework 1 and .NET 2, and for Axis 1.4. The WebSphere MQ custom channel for Microsoft Windows Communication Foundation runs services for .NET Framework 3. WebSphere Application Server and CICS provide support for running services over WebSphere MQ transport for SOAP. Create a custom Export to use WebSphere Enterprise Service Bus or WebSphere Process Server.

The WebSphere MQ SOAP listener can process SOAP requests transactionally. Run **amqwdeployWMQService** using the -x option. The two-phase option is only supported for listeners using server bindings. Other environments might provide transactional support for the WebSphere MQ transport for SOAP. Consult their documentation.

WebSphere MQ transport for SOAP currently does not support the emerging industry standard SOAP over JMS protocol that has been submitted to W3C. You can distinguish a SOAP/JMS message written to the new standard by looking for the JMS BindingVersion property. WebSphere MQ transport for SOAP does not set the BindingVersion property.

Axis 1.4

A Java class can typically be used without modification. The types of any arguments to the methods in the web service must be supported by the Axis engine. Refer to Axis documentation for further details. If the service uses a complex object as an argument, or returns one, that object must comply to the Java bean specification. See the examples in Figure 41 on page 2322, Figure 42 on page 2322, and Figure 43 on page 2322:

1. Have a public parameter-less constructor.
2. Any complex types of the bean must have public getters and setters of the form:

Prepare the service for deployment using the **amqwdeployWMQService** utility. The service is invoked by the WebSphere MQ SOAP listener which uses `axis.jar` to run the service.

The only two-phase transaction manager supported for Axis 1.4 is WebSphere MQ.

The supplied deployment utility does not support the case where a service returns an object in a different package to the service itself. To use an object returned in a different package, write your own deployment utility. You can base your deployment utility on the supplied sample, or capture the commands it produces, using the -v option. Amend the commands to produce a tailored script.

If the service uses classes that are external to the Axis infrastructure and the WebSphere MQ SOAP run time environment, you must set the correct `CLASSPATH`. To change `CLASSPATH`, amend the generated script that starts or defines the listeners to include the services required, in one of the following ways:

- Amend the `CLASSPATH` directly in the script after the call to **amqwsetcp**.
- Create a service-specific script to customize the `CLASSPATH` and invoke this script in the generated script after the call to **amqwsetcp**.
- Create a customized deployment process to customize the `CLASSPATH` in the generated script automatically.

.NET Framework 1 and .NET Framework 2

A service that has already been prepared as an HTTP Web service does not need to be modified for use as a WebSphere MQ Web service. It needs to be deployed using the **amqwdeployWMQService** utility.

The only two-phase transaction manager supported for .NET Framework 1 and .NET 2 is Microsoft Transaction Server (MTS).

If the service code has not been prepared as an HTTP Web service you must convert it to a Web service. Declare the class as a web service and identify how the parameters of each method are formatted. You

must check that any arguments to the methods of the service are compatible with the environment. Figure 39 and Figure 40 on page 2322 show a .NET class that has been prepared as a web service. The additions made are shown in bold type.

Figure 39 uses the code-behind programming model for a .NET Web service. In the code-behind model, the source for the service is separated from the `.asmx` file. The `.asmx` file declares the name of the associated source file with the Codebehind keyword. WebSphere MQ has samples of both inline and code-behind .NET Web services.

.NET Web services source must be compiled before deployment by the **amqwdeployWMQService** deployment utility. The service is compiled into a library (`.dll`). The library must be placed in the `./bin` subdirectory of the deployment directory.

.NET Framework 3

Create a WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF) to invoke services deployed to .NET Framework 3. See IBM WebSphere MQ custom channel for Microsoft Windows Communication Foundation (WCF) for a description of how to configure WCF to use the WebSphere MQ transport for SOAP.

WebSphere Application Server

You can invoke Web services hosted by WebSphere Application Server using the WebSphere MQ Transport for SOAP; see Using SOAP over JMS to transport Web services.

You need to modify the WSDL generated by deployment of a JMS service to WebSphere Application Server in order to generate a Web services client. The WSDL created by deployment to WebSphere Application Server includes a URI with a JNDI reference to the JMS `InitialContextFactory`. You need to modify the JNDI reference to Nojndi and provide connection attributes as described in "URI syntax and parameters for Web service deployment" on page 2307.

CICS

You can invoke CICS applications using the WebSphere MQ Transport for SOAP; see Configuring your CICS system for Web services.

WebSphere Enterprise Service Bus and WebSphere Process Server for Multiplatforms

WebSphere ESB and WebSphere Process Server for Multiplatforms support SOAP over JMS, with a ready built binding, only when using the default WebSphere Application Server messaging provider. Create a custom binding for JMS to support WebSphere MQ transport for SOAP. See JMS data bindings and Web services with SOAP over JMS in IBM WebSphere Process Server or IBM WebSphere Enterprise Service Bus, Part 2: Using the IBM WebSphere MQ JMS provider.

Example

```
<%@ WebService Language="C#" CodeBehind="Quote.asmx.cs" Class="Quote.QuoteDotNet" %>
```

Figure 39. Service definition for .NET Framework 2: `Quote.asmx`

```
<%@ WebService Language="C#" CodeBehind="Quote.asmx.cs" Class="Quote.QuoteDotNet" %>
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

namespace Quote {
    [WebService(Namespace = "http://www.example.org/")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    public class QuoteDotNet : System.Web.Services.WebService {
        [WebMethod]
        public string getQuote(String symbol){
            return symbol.ToUpper();
        }
    }
}
```

Figure 40. Service implementation for .NET Framework 2: `Quote.asmx.cs`

```
package org.example.www;
public interface CustomerInfoInterface extends java.rmi.Remote {
    public org.example.www.CustomerRecord
           getCustomerName(org.example.www.CustomerRecord request)
           throws java.rmi.RemoteException, org.example.www.GetCustomerName_faultMsg;
}
```

Figure 41. Java JAX-RPC service interface using a complex type

```
package org.example.www;
public class CustomerInfoPortImpl implements org.example.www.CustomerInfoInterface{
    public org.example.www.CustomerRecord
           getCustomerName(org.example.www.CustomerRecord request)
           throws java.rmi.RemoteException, org.example.www.GetCustomerName_faultMsg {
     request.setName(request.getID().toString());
     return request;
    }
}
```

Figure 42. Java JAX-RPC service implementation using a complex type

```
package org.example.www;
public class CustomerRecord  {
    private java.lang.String name;
    private java.lang.Integer ID;
    public CustomerRecord() {}
    public java.lang.String getName() {
        return name; }
    public void setName(java.lang.String name) {
        this.name = name; }
    public java.lang.Integer getID() {
        return ID; }
    public void setID(java.lang.Integer ID) {
        this.ID = ID; }
}
```

Figure 43. Java JAX-RPC service bean implementation of a complex type

WebSphere MQ transport for SOAP Web service clients

You can reuse an existing SOAP over HTTP client with WebSphere MQ transport for SOAP. You must make some small modifications to the code and build process to convert the client to work with WebSphere MQ transport for SOAP.

Coding

JAX-RPC clients must be written in Java. .NET Framework 1 and 2 clients can be written in any language that uses the Common Language Runtime. Code examples are provided in C# and Visual Basic.

The level of transactional support depends on the client environment and the pattern of the SOAP interaction. The SOAP request and SOAP reply can not be part of the same atomic transaction.

You must call `IBM.WMQSOAP.Register.Extension()` in a .NET Framework 1, .NET Framework 2 client. In a JAX-RPC Java Web service client call `com.ibm.mq.soap.Register.extension` to register the WebSphere MQ SOAP sender. The method registers the WebSphere MQ transport for SOAP sender as the handler for SOAP messages using the `jms:` protocol.

To create a .NET Framework 3 client, generate a Windows Communication Foundation client proxy using the **`svcutil`** tool; see Generating a WCF client proxy and application configuration files using the svcutil tool with metadata from a running service.

Libraries required to build and run .NET Framework 1 and 2 clients

- amqsoap
- System
- System.Web.Services
- System.Xml

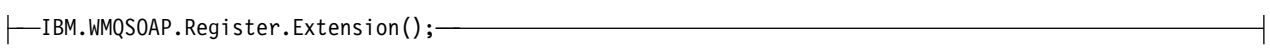
Libraries required to build and run Axis 1.4 clients

- *MQ_Install*\java\lib\com.ibm.mq.soap.jar;
- *MQ_Install*\java\lib\com.ibm.mq.commonservices.jar;
- *MQ_Install*\java\lib\soap\axis.jar;
- *MQ_Install*\java\lib\soap\jaxrpc.jar
- *MQ_Install*\java\lib\soap\saaj.jar;
- *MQ_Install*\java\lib\soap\commons-logging-1.0.4.jar;
- *MQ_Install*\java\lib\soap\commons-discovery-0.2.jar;
- *MQ_Install*\java\lib\soap\wsdl4j-1.5.1.jar;
- *MQ_Install*\java\jre\lib\xml.jar;
- *MQ_Install*\java\lib\soap\servlet.jar;
- *MQ_Install*\java\lib\com.ibm.mq.jar;
- *MQ_Install*\java\lib\com.ibm.mq.headers.jar;
- *MQ_Install*\java\lib\com.ibm.mq.pcf.jar;
- *MQ_Install*\java\lib\com.ibm.mq.jmqi.jar;
- *MQ_Install*\java\lib\com.ibm.mq.jmqi.remote.jar;
- *MQ_Install*\java\lib\com.ibm.mq.jmqi.local.jar;
- *MQ_Install*\java\lib\connector.jar;
- *MQ_Install*\java\lib\jta.jar;
- *MQ_Install*\java\lib\jndi.jar;
- *MQ_Install*\java\lib\ldap.jar

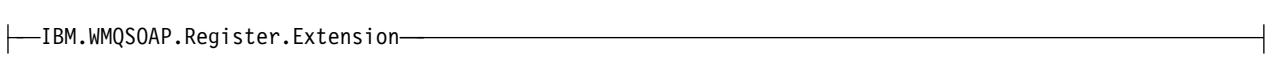
Register SOAP extension

Java:

C#:



Visual Basic:



Client examples

Figure 44 is an example of a .NET Framework 1 or .NET Framework 2 C# client that uses the inline programming model. The **IBM.WMQSOAP.Register.Extension()** method registers the WebSphere MQ SOAP sender with .NET as the `jms:` protocol handler.

```
using System;
namespace QuoteClientProgram {
    class QuoteMain {
        static void Main(string[] args) {
            try {
                IBM.WMQSOAP.Register.Extension();
                Quote q = new Quote();
                Console.WriteLine("Response is: " + q.getQuote("ibm"));
             }  catch (Exception e) {
                Console.WriteLine("Exception is: " + e);
             }
        }
    }
}
```

Figure 44. C# Web service client sample

Figure 45 on page 2325 is an example of a Java client that uses the JAX-RPC static proxy client interface. The **com.ibm.mq.soap.Register.extension();** method registers the WebSphere MQ SOAP sender with the service proxy to handle the `jms:` protocol.

```
package org.example.www;
import com.ibm.mq.soap.Register;
public class QuoteClient {
   public static void main(String[] args) {
      try {
         Register.extension();
         QuoteSOAPImplServiceLocator locator = new QuoteSOAPImplServiceLocator();
         System.out.println("Response = "
                  + locator.getOrgExampleWwwQuoteSOAPImpl_Wmq().getQuote("IBM"));
      } catch (Exception e) {
         System.out.println("Exception = " + e.getMessage());
      }
   }
}
```

Figure 45. Java Web service client example

User exits, API exits, and installable services reference

Use the links provided in this section to help you develop your User exits, API exits, and installable services applications:

- "MQIEP structure"
- "Data-conversion exit reference" on page 2329
- "MQ_PUBLISH_EXIT - Publish exit" on page 2333
- "Channel-exit calls and data structures" on page 2341
- "API exit reference" on page 2413
- "Installable services interface reference information" on page 2476

Related concepts:

"MQI applications reference" on page 1337
Use the links provided in this section to help you develop your MQI applications:

"The WebSphere MQ classes for Java libraries" on page 2748
The location of the WebSphere MQ classes for Java libraries varies according to platform. Specify this location when you start an application.

Related reference:

"SOAP reference" on page 2273
WebSphere MQ transport for SOAP reference information arranged alphabetically.

"Reference material for WebSphere MQ bridge for HTTP" on page 2542
Reference topics for WebSphere MQ bridge for HTTP, arranged alphabetically

"The WebSphere MQ .NET classes and interfaces" on page 2578
WebSphere MQ .NET classes and interfaces are listed alphabetically. The properties, methods and constructors are described.

"WebSphere MQ C++ classes" on page 2639
The WebSphere MQ C++ classes encapsulate the WebSphere MQ Message Queue Interface (MQI). There is a single C++ header file, **imqi.hpp**, which covers all of these classes.

Related information:

Developing applications

WebSphere MQ Classes for JMS

MQIEP structure

The MQIEP structure contains an entry point for each function call that exits are permitted to make.

Fields

StrucId
Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQIEP_STRUC_ID

Version
Type: MQLONG - input

Structure version number. The value is as follows:

MQIEP_VERSION_1
Version 1 structure version number.

MQIEP_CURRENT_VERSION
Current version of the structure.

StrucLength
Type: MQLONG

Size of the MQIEP structure in bytes. The value is as follows:

MQIEP_LENGTH_1

Flags
Type: MQLONG

Provides information about the function addresses. A flag to indicate if the library is threaded can be used with a flag to indicate if the library is a client or server library.

The following value is used to specify no library information:

MQIEPF_NONE

One of the following values is used to specify if the shared library is threaded or non-threaded:

MQIEPF_NON_THREADED_LIBRARY
A non-threaded shared library

MQIEPF_THREADED_LIBRARY
A threaded shared library

One of the following values is used to specify if the shared library is a client or a server shared library:

MQIEPF_CLIENT_LIBRARY
A client shared library

MQIEPF_LOCAL_LIBRARY
A server shared library

Reserved
Type: MQPTR

MQBACK_Call
Type: PMQ_BACK_CALL

Address of the MQBACK call.

MQBEGIN_Call
Type: PMQ_BEGIN_CALL

Address of the MQBEGIN call.

MQBUFMH_Call
Type: PMQ_BUFMH_CALL

Address of the MQBUFMH call.

MQCB_Call
Type: PMQ_CB_CALL

Address of the MQCB call.

MQCLOSE_Call
Type: PMQ_CLOSE_CALL

Address of the MQCLOSE call.

MQCMIT_Call
Type: PMQ_CMIT_CALL

Address of the MQCMIT call.

MQCONN_Call
Type: PMQ_CONN_CALL

Address of the MQCONN call.

MQCONNX_Call
Type: PMQ_CONNX_CALL

Address of the MQCONNX call.

MQCRTMH_Call
Type: PMQ_CRTMH_CALL

Address of the MQCRTMH call.

MQCTL_Call
Type: PMQ_CTL_CALL

Address of the MQCTL call.

MQDISC_Call
Type: PMQ_DISC_CALL

Address of the MQDISC call.

MQDLTMH_Call
Type: PMQ_DLTMH_CALL

Address of the MQDLTMH call.

MQDLTMP_Call
Type: PMQ_DLTMP_CALL

Address of the MQDLTMP call.

MQGET_Call
Type: PMQ_GET_CALL

Address of the MQGET call.

MQINQ_Call
Type: PMQ_INQ_CALL

Address of the MQINQ call.

MQINQMP_Call
Type: PMQ_INQMP_CALL

Address of the MQINQMP call.

MQMHBUF_Call
Type: PMQ_MHBUF_CALL

Address of the MQMHBUF call.

MQOPEN_Call
Type: PMQ_OPEN_CALL

Address of the MQOPEN call.

MQPUT_Call
Type: PMQ_PUT_CALL

Address of the MQPUT call.

MQPUT1_Call
Type: PMQ_PUT1_CALL

Address of the MQPUT1 call.

MQSET_Call
Type: PMQ_SET_CALL

Address of the MQSET call.

MQSETMP_Call
Type: PMQ_SETMP_CALL

Address of the MQSETMP call.

MQSTAT_Call
Type: PMQ_STAT_CALL

Address of the MQSTAT call.

MQSUB_Call
Type: PMQ_SUB_CALL

Address of the MQSUB call.

MQSUBRQ_Call
Type: PMQ_SUBRQ_CALL

Address of the MQSUBRQ call.

MQXCNVC_Call
Type: PMQ_XCNVC_CALL

Address of the MQXCNVC call.

MQXCLWLN_Call
Type: PMQ_XCLWLN_CALL

Address of the MQXCLWLN call.

MQXDX_Call
Type: PMQ_XDX_CALL

Address of the MQXDX call.

MQXEP_Call
Type: PMQ_XEP_CALL

Address of the MQXEP call.

MQZEP_Call
Type: PMQ_ZEP_CALL

Address of the MQZEP call.

C Declaration

```
struct tagMQIEP {
   MQCHAR4          StrucId;          /* Structure identifier */
   MQLONG           Version;          /* Structure version number */
   MQLONG           StrucLength;      /* Structure length */
   MQLONG           Flags;            /* Flags */
   MQPTR            Reserved;         /* Reserved */
   PMQ_BACK_CALL    MQBACK_Call;      /* Address of MQBACK */
   PMQ_BEGIN_CALL   MQBEGIN_Call;     /* Address of MQBEGIN */
   PMQ_BUFMH_CALL   MQBUFMH_Call;     /* Address of MQBUFMH */
   PMQ_CB_CALL      MQCB_Call;        /* Address of MQCB */
   PMQ_CLOSE_CALL   MQCLOSE_Call;     /* Address of MQCLOSE */
   PMQ_CMIT_CALL    MQCMIT_Call;      /* Address of MQCMIT */
   PMQ_CONN_CALL    MQCONN_Call;      /* Address of MQCONN */
   PMQ_CONNX_CALL   MQCONNX_Call;     /* Address of MQCONNX */
   PMQ_CRTMH_CALL   MQCRTMH_Call;     /* Address of MQCRTMH */
   PMQ_CTL_CALL     MQCTL_Call;       /* Address of MQCTL */
   PMQ_DISC_CALL    MQDISC_Call;      /* Address of MQDISC */
   PMQ_DLTMH_CALL   MQDLTMH_Call;     /* Address of MQDLTMH */
   PMQ_DLTMP_CALL   MQDLTMP_Call;     /* Address of MQDLTMP */
   PMQ_GET_CALL     MQGET_Call;       /* Address of MQGET */
   PMQ_INQ_CALL     MQINQ_Call;       /* Address of MQINQ */
   PMQ_INQMP_CALL   MQINQMP_Call;     /* Address of MQINQMP */
   PMQ_MHBUF_CALL   MQMHBUF_Call;     /* Address of MQMHBUF */
   PMQ_OPEN_CALL    MQOPEN_Call;      /* Address of MQOPEN */
   PMQ_PUT_CALL     MQPUT_Call;       /* Address of MQPUT */
   PMQ_PUT1_CALL    MQPUT1_Call;      /* Address of MQPUT1 */
   PMQ_SET_CALL     MQSET_Call;       /* Address of MQSET */
   PMQ_SETMP_CALL   MQSETMP_Call;     /* Address of MQSETMP */
   PMQ_STAT_CALL    MQSTAT_Call;      /* Address of MQSTAT */
   PMQ_SUB_CALL     MQSUB_Call;       /* Address of MQSUB */
   PMQ_SUBRQ_CALL   MQSUBRQ_Call;     /* Address of MQSUBRQ */
   PMQ_XCLWLN_CALL  MQXCLWLN_Call;    /* Address of MQXCLWLN */
   PMQ_XCNVC_CALL   MQXCNVC_Call;     /* Address of MQXCNVC */
   PMQ_XDX_CALL     MQXDX_Call;       /* Address of MQXDX */
   PMQ_XEP_CALL     MQXEP_Call;       /* Address of MQXEP */
   PMQ_ZEP_CALL     MQZEP_Call;       /* Address of MQZEP */
 };
```

Data-conversion exit reference

For z/OS, you must write data-conversion exits in assembler language. For other platforms, it is recommended that you use the C programming language.

To help you to create a data-conversion exit program, the following are supplied:

- A skeleton source file
- A convert characters call
- A utility that creates a fragment of code that performs data conversion on data type structures This utility takes C input only. On z/OS, it produces assembler code.

For the procedure for writing the programs see:

- Writing a data-conversion exit for WebSphere MQ on UNIX and Linux systems
- Writing a data-conversion exit for WebSphere MQ for Windows

Skeleton source file:

These can be used as your starting point when writing a data-conversion exit program.

The files supplied are listed in Table 234.

Table 234. Skeleton source files

Platform	File
AIX	amqsvfc0.c
IBM i	QMQMSAMP/QCSRC(AMQSVFC4)
HP-UX	amqsvfc0.c
Linux	amqsvfc0.c
z/OS	CSQ4BAX8 (1) CSQ4BAX9 (2) CSQ4CAX9 (3)
Solaris	amqsvfc0.c
Windows systems	amqsvfc0.c
Notes: 1. Illustrates the MQXCVNC call. 2. A wrapper for the code fragments generated by the utility for use in all environments except CICS. 3. A wrapper for the code fragments generated by the utility for use in the CICS environment.	

Convert characters call:

Use the MQXCNVC (convert characters) call from within a data-conversion exit program to convert character message data from one character set to another. For certain multibyte character sets (for example, UCS2 character sets), the appropriate options must be used.

No other MQI calls can be made from within the exit; an attempt to make such a call fails with reason code MQRC_CALL_IN_PROGRESS.

See “MQXCNVC - Convert characters” on page 2223 for further information on the MQXCNVC call and appropriate options.

Utility for creating conversion-exit code:

Use this information to learn more about creating conversion-exit code.

The commands for creating conversion-exit code are:

IBM i CVTMQMDTA (Convert WebSphere MQ Data Type)

Windows, UNIX and Linux systems
`crtmqcvx` (Create WebSphere MQ conversion-exit)

The command for your platform produces a fragment of code that performs data conversion on data type structures, for use in your data-conversion exit program. The command takes a file containing one or more C language structure definitions. .

Error messages in Windows, UNIX and Linux systems

The `crtmqcvx` command returns messages in the range AMQ7953 through AMQ7970.

These messages are listed in *WebSphere MQ Messages*.

There are two main types of error:

- Major errors, such as syntax errors, when processing cannot continue.

  A message is displayed on the screen giving the line number of the error in the input file. The output file might have been partially created.
- Other errors when a message is displayed stating that a problem has been found but that parsing of the structure can continue.

  The output file has been created and contains error information about the problems that have occurred. This error information is prefixed by `#error` so that the code produced is not accepted by any compiler without intervention to rectify the problems.

Valid syntax:

Your input file for the utility must conform to the C language syntax.

If you are unfamiliar with C, refer to the C example in this topic.

In addition, be aware of the following rules:

- typedef is recognized only before the struct keyword.
- A structure tag is required on your structure declarations.
- You can use empty square brackets [ ] to denote a variable length array or string at the end of a message.
- Multidimensional arrays and arrays of strings are not supported.
- The following additional data types are recognized:
  - MQBOOL
  - MQBYTE
  - MQCHAR
  - MQFLOAT32
  - MQFLOAT64
  - MQSHORT
  - MQLONG
  - MQINT8
  - MQUINT8
  - MQINT16
  - MQUINT16
  - MQINT32
  - MQUINT32
  - MQINT64
  - MQUINT64

  MQCHAR fields are code page converted, but MQBYTE, MQINT8 and MQUINT8 are left untouched. If the encoding is different, MQSHORT, MQLONG, MQINT16, MQUINT16, MQINT32, MQUINT32, MQINT64, MQUINT64, MQFLOAT32, MQFLOAT64 and MQBOOL are converted accordingly.
- Do *not* use the following types of data:
  - double

– pointers
– bit-fields

This is because the utility for creating conversion-exit code does not provide the facility to convert these data types. To overcome this, you can write your own routines and call them from the exit.

Other points to note:

- Do not use sequence numbers in the input data set.
- If there are fields for which you want to provide your own conversion routines, declare them as MQBYTE, and then replace the generated CMQXCFBA macros with your own conversion code.

C example

```
struct TEST { MQLONG    SERIAL_NUMBER;
              MQCHAR    ID[5];
              MQINT16   VERSION;
              MQBYTE    CODE[4];
              MQLONG    DIMENSIONS[3];
              MQCHAR    NAME[24];
            } ;
```

This corresponds to the following declarations in other programming languages:

COBOL

```
      10 TEST.
       15 SERIAL-NUMBER  PIC S9(9) BINARY.
       15 ID             PIC X(5).
       15 VERSION        PIC S9(4) BINARY.
     * CODE IS NOT TO BE CONVERTED
       15 CODE           PIC X(4).
       15 DIMENSIONS     PIC S9(9) BINARY OCCURS 3 TIMES.
       15 NAME           PIC X(24).
```

System/390

```
TEST           EQU *
SERIAL_NUMBER  DS  F
ID             DS  CL5
VERSION        DS  H
CODE           DS  XL4
DIMENSIONS     DS  3F
NAME           DS  CL24
```

PL/I

Supported on z/OS only

```
 DCL 1 TEST,
       2 SERIAL_NUMBER  FIXED BIN(31),
       2 ID             CHAR(5),
       2 VERSION        FIXED BIN(15),
       2 CODE           CHAR(4),        /* not to be converted */
       2 DIMENSIONS(3)  FIXED BIN(31),
       2 NAME           CHAR(24);
```

MQ_PUBLISH_EXIT - Publish exit

The MQ_PUBLISH_EXIT call can inspect and alter messages delivered to subscribers.

Purpose

Use the publish exit to inspect and alter messages delivered to subscribers:

- Examine the contents of a message published to each subscriber
- Modify the contents of a message published to each subscriber
- Alter the queue to which a message is put
- Stop the delivery of a message to a subscriber

Syntax

MQ_PUBLISH_EXIT(*ExitParms*, *PubContext*, *SubContext*)

Parameters

ExitParms **(MQPSXP)- Input/Output**
ExitParms contains information about the invocation of the exit.

PubContext **(MQPBC) - Input**
PubContext contains contextual information about the publisher of the publication.

SubContext **(MQSBC) - Input/Output**
SubContext contains contextual information about the subscriber receiving the publication.

MQPSXP - Publish exit data structure:

The MQPSXP structure describes the information that is passed to and returned from the publish exit.

Table 235 summarizes the fields in the structure:

Table 235. Fields in MQPSXP

Field	Description
StrucID	Structure identifier
Version	Structure version number
ExitId	Type of exit that is being called
ExitReason	Reason for calling the exit
ExitResponse	Response from the exit
ExitResponse2	Secondary response from exit
Feedback	Feedback code
ExitUserArea	Exit user area
ExitData	Exit data
QMgrName	Name of local queue manager
Hconn	Connection handle
MsgDescPtr	Address of message descriptor (MQMD)
MsgHandle	Handle to message properties (MQHMSG)
MsgInPtr	Address of input message
MsgInLength	Length of input message
MsgOutPtr	Address of output message

Table 235. Fields in MQPSXP (continued)

Field	Description
MsgOutLength	Length of output message
pEntryPoints	Address of the MQIEP structure

Fields

StrucID **(MQCHAR4)**

StrucID is the structure identifier. The value is as follows:

MQPSXP_STRUCID
MQPSXP_STRUCID is the identifier for the publish exit parameter structure. For the C programming language, the constant MQPSXP_STRUC_ID_ARRAY is also defined; it has the same value as MQPSXP_STRUC_ID, but is an array of characters instead of a string.

StrucID is an input field to the exit.

Version **(MQLONG)**

Version is the structure version number. The value is as follows:

MQPSXP_VERSION_1
MQPSXP_VERSION_1 is the Version 1 publish exit parameter structure. The constant MQPSXP_CURRENT_VERSION is also defined with the same value.

Version is an input field to the exit.

ExitId **(MQLONG)**

ExitId is the type of exit that is being called. The value is as follows:

MQXT_PUBLISH_EXIT
Publish exit.

ExitId is an input field to the exit.

ExitReason **(MQLONG)**

ExitReason is the reason for calling the exit. The possible values are:

MQXR_INIT
The exit for this connection is called for initialization. The exit might acquire and initialize the resources that it needs; for example, main storage.

MQXR_TERM
The exit for this connection is called because the exit is about to be stopped. The exit must free any resources that it has acquired since it was initialized; for example, main storage.

MQXR_PUBLICATION
The exit is called by the queue manager before it puts a publication onto a message queue of a subscriber. The exit can change the message, not put the message on the queue, or halt publication.

ExitReason is an input field to the exit.

ExitResponse **(MQLONG)**

Set *ExitResponse* in the exit to specify how processing must continue. *ExitResponse* is one of the following values:

MQXCC_OK
Set MQXCC_OK to continue processing normally. Set MQXCC_OK in response to any values of ExitReason.

If ExitReason has the value MQXR_PUBLICATION , the *DestinationQName* and *DestinationQMgrName* fields of the MQSBC structure identify the destination to which the message is sent.

MQXCC_FAILED
Set MQXCC_FAILED to stop the publish operation. The completion code MQCC_FAILED and reason code 2557 (09FD) (RC2557): MQRC_PUBLISH_EXIT_ERROR is set on return from the exit.

MQXCC_SUPPRESS_FUNCTION
Set MQXCC_SUPPRESS_FUNCTION to stop normal processing of the message. Only set MQXCC_SUPPRESS_FUNCTION if *ExitReason* has the value MQXR_PUBLICATION .

The message continues to be processed by the queue manager according to the MQRO_DISCARD_MSG option in the *Report* field in the message descriptor of the message.

- If the MQRO_DISCARD_MSG option is specified, the message is not delivered to the subscriber.
- If the MQRO_DISCARD_MSG option is not specified, the message is placed on the dead-letter queue. If there is no dead-letter queue, or the message cannot be placed successfully on the dead-letter queue, the publication is not delivered to the subscriber. The delivery of the publication to other subscribers depends on the values of the PMSGDLV and NPMSGDLV topic object attributes. For an explanation of these attributes, see the parameter descriptions for the DEFINE TOPIC command.

ExitResponse is an output field from the exit.

ExitResponse2 **(MQLONG)**
ExitResponse2 is reserved for future use.

Feedback **(MQLONG)**
Feedback is the feedback code to be used if the exit returns MQXCC_SUPPRESS_FUNCTION in *ExitResponse*.

On input to the exit, *Feedback* always has the value MQFB_NONE. If the exit returns MQXCC_SUPPRESS_FUNCTION, set *Feedback* to the value to be used for the message when the queue manager places it on the dead-letter queue. On return from the exit, if *Feedback* has the original value MQFB_NONE, the queue manager sets *Feedback* to MQFB_STOPPED_BY_PUBSUB_EXIT.

Feedback is an input/output field to the exit.

ExitUserArea **(MQBYTE16)**
ExitUserArea is a field that is available for the exit to use. Each connection has a separate *ExitUserArea*. The length of *ExitUserArea* is given by MQ_EXIT_USER_AREA_LENGTH .

The *ExitReason* field has the value MQXR_INIT on the first invocation of the exit. *ExitUserArea* is initialized to MQXUA_NONE on the first invocation of the exit for a connection. Subsequent changes to *ExitUserArea* are preserved across invocations of the exit.

ExitUserArea is an input/output field to the exit.

ExitData **(MQCHAR32)**
ExitData is fixed exit data defined by the *PublishExitData* parameter of the stanza in the initialization file of the queue manager. The data is padded with blanks to the full length of the field. If there is no fixed exit data defined in the initialization file, *ExitData* is blank. The length of *ExitData* is given by MQ_EXIT_DATA_LENGTH.

ExitData is an input field to the exit.

QMgrName **(MQCHAR48)**
QMgrName is the name of the local queue manager. The name is padded with blanks to the full length of the field. The length of this field is given by MQ_Q_MGR_NAME_LENGTH .

QMgrName is an input field to the exit.

Hconn **(MQHCONN)**
Hconn is the handle representing a connection to the queue manager. Only use *Hconn* as a parameter to the MQSETMP, MQINQMMP, or MQDLTMP message property function calls to work with message properties.

Hconn is an input field to the exit.

MsgDescPtr **(PMQMD)**
MsgDescPtr is the address of message descriptor (MQMD) of the message being processed, and is a copy of the MQMD returned from the MQPUT call. The exit can change the contents of the message descriptor. Any change to the contents of the message descriptor must be done with care. In particular, in the case where the *SubType* field of the MQSBC structure is of value MQSUBTYPE_PROXY, the *CorrelId* field in the message descriptor must not be changed.

No message descriptor is passed to the exit if *ExitReason* is MQXR_INIT or MQXR_TERM ; in these cases, *MsgDescPtr* is the null pointer.

MsgDescPtr is an input field to the exit.

MsgHandle **(MQHMSG)**
MsgHandle is the handle to message properties. Only use *MsgHandle* with the MQSETMP, MQINQMMP , or MQDLTMP message property function calls to work with message properties.

MsgHandle is an input field to the exit.

MsgInPtr **(PMQVOID)**
MsgInPtr is the address of the input message data. The contents of the buffer addressed by *MsgInPtr* can be modified by the exit; see *MsgOutPtr* .

MsgInPtr is an input field to the exit.

MsgInLength **(MQLONG)**
MsgInLength is the length in bytes of the message data passed to the exit. The address of the data is given by *MsgInPtr*.

MsgInLength is an input field to the exit.

MsgOutPtr **(PMQVOID)**
MsgOutPtr is the address of a buffer containing message data that is returned from the exit. On entry to the exit, *MsgOutPtr* is null. On return from the exit, if the value is still null, the queue manager sends the message specified by *MsgInPtr* , with the length given by *MsgInLength* .

If the exit modifies the message data, use one of the following procedures:

- If the length of the data does not change, the data can be modified in the buffer addressed by *MsgInPtr* . In this case, do not change *MsgOutPtr* and *MsgOutLength*.
- If the modified data is shorter than the original data, the data can be modified in the buffer addressed by *MsgInPtr* . In this case *MsgOutPtr* must be set to the address of the input message buffer, and *MsgOutLength* set to the new length of the message data.
- If the modified data is, or might be, longer than the original data, the exit must obtain a new message buffer. Copy the modified data into it. Set *MsgOutPtr* to the address of the new buffer, and set *MsgOutLength* to the length of the new message data. The exit is responsible for freeing the buffer addressed by *MsgOutPtr* when the exit is next called.

Note: *MsgOutPtr* is always the null pointer on input to the exit, and not the address of a previously obtained message buffer. To free the previously obtained buffer, the exit must save its address and length. Save the information either in *ExitUserArea*, or in a control block that has its address saved in *ExitUserArea* .

MsgOutPtr is an input/output field to the exit.

MsgOutLength **(MQLONG)**
MsgOutLength is the length in bytes of the message data returned by the exit. On input to the exit,

this field is always zero. On return from the exit, this field is ignored if *MsgOutPtr* is null. See *MsgOutPtr* for information about modifying the message data.

MsgOutLength is an input/output field to the exit.

pEntryPoints **(PMQIEP)**
pEntryPoints is the address of an MQIEP structure through which MQI and DCI calls can be made.

C language declaration - MQPSXP

```
typedef struct tagMQPSXP {
 MQCHAR4    StrucId;             /* Structure identifier */
 MQLONG     Version;             /* Structure version number */
 MQLONG     ExitId;              /* Type of exit */
 MQLONG     ExitReason;          /* Reason for invoking exit */
 MQLONG     ExitResponse;        /* Response from exit */
 MQLONG     ExitResponse2;       /* Reserved */
 MQLONG     Feedback;            /* Feedback code */
 MQBYTE16   ExitUserArea;        /* Exit user area */
 MQCHAR32   ExitData;            /* Exit data */
 MQCHAR48   QMgrName;            /* Name of local queue manager */
 MQHCONN    Hconn;               /* Connection handle */
 MQHMSG     MsgHandle;           /* Handle to message properties */
 PMQMD      MsgDescPtr;          /* Address of message descriptor */
 PMQVOID    MsgInPtr;            /* Address of input message data */
 MQLONG     MsgInLength;         /* Length of input message data */
 PMQVOID    MsgOutPtr;           /* Address of output message data */
 MQLONG     MsgOutLength;        /* Length of output message data */
 /* Ver:1 */
 PMQIEP     pEntryPoints;        /* Address of the MQIEP structure */
 /* Ver:2 */
} MQPSXP;
```

MQPBC - Publication context data structure:

The MQPBC structure contains the contextual information, relating to the publisher of the publication, that is passed to the publish exit.

Table 236 summarizes the fields in the structure:

Table 236. Fields in MQPBC

Field	Description
StrucID	Structure identifier
Version	Structure version number
PubTopicString	Publish topic string
MsgDescPtr	Address of message descriptor (MQMD)

Fields

StrucID **(MQCHAR4)**
StrucID is the structure identifier. The value is as follows:

MQPBC_STRUCID
MQPBC_STRUCID is the identifier for the publication context structure. For the C programming language, the constant MQPBC_STRUC_ID_ARRAY is also defined; it has the same value as MQPBC_STRUC_ID, but is an array of characters instead of a string.

StrucID is an input field to the exit.

Version **(MQLONG)**
Version is the structure version number. The value is as follows:

MQPBC_VERSION_1
MQPBC_VERSION_1 is the Version 1 publish exit parameter structure.

MQPBC_VERSION_2
MQPBC_VERSION_2 is the Version 2 publish exit parameter structure. The constant MQPBC_CURRENT_VERSION is also defined with the same value.

Version is an input field to the exit.

PubTopicString **(MQCHARV)**
PubTopicString is the topic string being published to.

PubTopicString is an input field to the exit.

MsgDescPtr **(PMQMD)**
MsgDescPtr is the address of a copy of the message descriptor (MQMD) for the message being processed.

MsgDescPtr is an input field to the exit.

C language declaration - MQPBC

```
typedef struct tagMQPBC {
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQCHARV    PubTopicString;      /* Publish topic string */
  PMQMD      MsgDescPtr;          /* Address of message descriptor */
} MQPBC;
```

MQSBC - Subscription context data structure:

The MQSBC structure contains the contextual information, relating to the subscriber that is receiving the publication, that is passed to the publish exit.

Table 237 summarizes the fields in the structure:

Table 237. Fields in MQSBC

Field	Description
StrucID	Structure identifier
Version	Structure version number
DestinationQMgrName	Name of destination queue manager
DestinationQName	Name of destination queue
SubType	Type of subscription
SubOptions	Subscription options
ObjectName	Object name
ObjectString	Object string
SubTopicString	Subscription topic string
SubName	Subscription name
SubId	Subscription identifier
SelectionString	Address of selection string
SubLevel	Subscription level
PSProperties	Publish/subscribe properties

Fields

StrucID **(MQCHAR4)**
Structure identifier. The value is as follows:

MQSBC_STRUCID
MQSBC_STRUCID is the identifier for the publish exit parameter structure. For the C programming language, the constant MQSBC_STRUC_ID_ARRAY is also defined; MQSBC_STRUC_ID_ARRAY has the same value as MQSBC_STRUC_ID, but is an array of characters instead of a string.

StrucID is an input field to the exit.

Version **(MQLONG)**
Structure version number. The value is as follows:

MQSBC_VERSION_1
Version 1 publish exit parameter structure. The constant MQSBC_CURRENT_VERSION is also defined with the same value.

Version is an input field to the exit.

DestinationQMgrName **(MQCHAR48)**
DestinationQMgrName is the name of the queue manager to which the message is being sent. The name is padded with blanks to the full length of the field. The name can be altered by the exit. The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

DestinationQMgrName is an input/output field to the exit; see note.

DestinationQName **(MQCHAR48)**
DestinationQName is the name of the queue to which the message is being sent. The name is padded with blanks to the full length of the field. The name can be altered by the exit. The length of this field is given by MQ_Q_NAME_LENGTH.

DestinationQName is an input/output field to the exit; see note.

SubType **(MQLONG)**
SubType indicates how the subscription was created. Valid values are MQSUBTYPE_API, MQSUBTYPE_ADMIN and MQSUBTYPE_PROXY; see Inquire Subscription Status (Response).

SubType is an input field to the exit.

SubOptions **(MQLONG)**
SubOptions are the subscription options; see "Options (MQLONG)" on page 1863 for a description of values this field can take.

SubOptions is an input field to the exit.

ObjectName **(MQCHAR48)**
ObjectName is the name of the topic object as defined on the local queue manager. The length of this field is given by MQ_TOPIC_NAME_LENGTH. The object name is the name of the administrative topic object that the queue manager has associated with the topic string. Even if the subscriber provided a topic object as part of the subscription, the *ObjectName* might be a different topic object. The association of a topic object with a subscription depends upon the full resolution of *SubTopicString*.

ObjectName is an input field to the exit.

ObjectString **(MQCHARV)**
ObjectString is the full topic string of the publication that was subscribed to. Any wildcards in the original subscription string are resolved. It is different to the MQSD subscription *ObjectString* field described in "ObjectString (MQCHARV)" on page 1863, which might contain wildcards, and is exclusive of any object name provided by the subscriber.

ObjectString is an input field to the exit.

SubTopicString **(MQCHARV)**
SubTopicString is the complete topic string as supplied by the subscriber. *SubTopicString* is the combination of the topic string defined in a topic object, and a topic string. A subscriber must provide either a topic object, a topic string, or both. If the subscriber provides a topic string, it might contain wildcards.

SubTopicString is an input field to the exit.

SubName **(MQCHARV)**
SubName is the subscription name that is provided either by the subscriber, or is a generated name.

SubName is an input field to the exit.

SubId **(MQBYTE 24)**
SubId is the unique internal subscription identifier.

SubId is an input field to the exit.

SelectionString **(MQCHARV)**
SelectionString is the selection criteria used when subscribing for messages from a topic; seeSelectors.

SelectionString is an input field to the exit.

SubLevel **(MQLONG)**
SubLevel is the interception level associated with the subscription; see "SubLevel (MQLONG)" on page 1875 for further details.

SubLevel is an input field to the exit.

PSProperties **(MQLONG)**
PSProperties are the publish/subscribe properties. They specify how publish/subscribe related message properties are added to messages sent to this subscription. Possible values are MQPSPROP_NONE, MQPSPROP_COMPAT, MQPSPROP_RFH2, MQPSPROP_MSGPROP. See Optional parameters (Change, Copy, and Create Subscription) for a description of these values.

PSProperties is an input field to the exit.

Note: Authorization checks are only performed on the original values of *DestinationQMgrName* and *DestinationQName* before they are passed to the publish exit. No new authorization checks are performed when the exit changes the destination queue, either by changing *DestinationQMgrName* or *DestinationQName*.

C language declaration - MQSBC

```
typedef struct tagMQSBC {
  MQCHAR4    StrucId;             /* Structure identifier */
  MQLONG     Version;             /* Structure version number */
  MQCHAR48   DestinationQMgrName; /* Destination queue manager */
  MQCHAR48   DestinationQName;    /* Destination queue name */
  MQLONG     SubType;             /* Type of subscription */
  MQLONG     SubOptions;          /* Subscription options */
  MQCHAR48   ObjectName;          /* Object name */
  MQCHARV    ObjectString;        /* Object string */
  MQCHARV    SubTopicString;      /* Subscription topic string */
  MQCHARV    SubName;             /* Subscription name */
  MQBYTE24   SubId;               /* Subscription identifier */
  MQCHARV    SelectionString;     /* Subscription selection string */
  MQLONG     SubLevel;            /* Subscription level */
  MQLONG     PSProperties;        /* Publish/subscribe properties */
} MQSBC;
```

Channel-exit calls and data structures

This collection of topics provide reference information about the special WebSphere MQ calls and data structures that you can use when you write channel exit programs.

This information is product-sensitive programming interface information. You can write WebSphere MQ user exits in the following programming languages:

Platform	Programming languages
WebSphere MQ for z/OS	Assembler and C (which must conform to the C system programming environment for system exits, described in the *z/OS C/C++ Programming Guide*.)
WebSphere MQ for IBM i	ILE C, ILE COBOL, and ILE RPG
All other WebSphere MQ platforms	C

You can also write user exits in Java for use only with Java and JMS applications. For more information about creating and using channel exits with the WebSphere MQ classes for Java, see Using channel exits in WebSphere MQ classes for Java and for WebSphere MQ classes for JMS, see Using channel exits with WebSphere MQ classes for JMS.

You cannot write WebSphere MQ user exits in TAL or Visual Basic. However, a declaration for the MQCD structure is provided in Visual Basic for use on the MQCONNX call from an WebSphere MQ MQI client program.

In a number of cases in the descriptions that follow, parameters are arrays or character strings with a size that is not fixed. For these parameters, a lowercase "n" is used to represent a numeric constant. When the declaration for that parameter is coded, the "n" must be replaced by the numeric value required. For further information about the conventions used in these descriptions, see the "Elementary data types" on page 1530.

Data definition files

Data definition files are supplied with WebSphere MQ for each of the supported programming languages. For details of these files, see Copy, header, include, and module files.

MQ_CHANNEL_EXIT - Channel exit:

The MQ_CHANNEL_EXIT call describes the parameters that are passed to each of the channel exits called by the Message Channel Agent.

No entry point called MQ_CHANNEL_EXIT is provided by the queue manager; the name MQ_CHANNEL_EXIT is of no special significance since the names of the channel exits are provided in the channel definition MQCD.

There are five types of channel exit:

- Channel security exit
- Channel message exit
- Channel send exit
- Channel receive exit
- Channel message-retry exit

The parameters are similar for each type of exit, and the description given here applies to all of them, except where specifically noted.

Syntax

MQ_CHANNEL_EXIT *(ChannelExitParms, ChannelDefinition, DataLength, AgentBufferLength, AgentBuffer, ExitBufferLength, ExitBufferAddr)*

Parameters

The MQ_CHANNEL_EXIT call has the following parameters.

ChannelExitParms (MQCXP) - input/output

Channel exit parameter block.

This structure contains additional information relating to the invocation of the exit. The exit sets information in this structure to indicate how the MCA proceeds.

ChannelDefinition (MQCD) - input/output

Channel definition.

This structure contains parameters set by the administrator to control the behavior of the channel.

DataLength (MQLONG) - input/output

Length of data.

The data depends on the type of exit:

- For a channel security exit, when the exit is invoked this parameter contains the length of any security message in the *AgentBuffer* field, if *ExitReason* is MQXR_SEC_MSG. It is zero if there is no message. The exit must set this field to the length of any security message to be sent to its partner if it sets *ExitResponse* to MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG. The message data is in either *AgentBuffer* or *ExitBufferAddr*.

  The content of security messages is the sole responsibility of the security exits.
- For a channel message exit, when the exit is invoked this parameter contains the length of the message (including the transmission queue header). The exit must set this field to the length of the message in either *AgentBuffer* or *ExitBufferAddr* that is to proceed. This must be greater than or equal to the length of the transmission queue header (MQXQH).
- For a channel send or channel receive exit, when the exit is invoked this parameter contains the length of the transmission. The exit must set this field to the length of the transmission in either *AgentBuffer* or *ExitBufferAddr* that is to proceed.

If a security exit sends a message, and there is no security exit at the other end of the channel, or the other end sets an *ExitResponse* of MQXCC_OK, the initiating exit is re-invoked with MQXR_SEC_MSG and a null response (*DataLength*=0).

AgentBufferLength (MQLONG) - input

Length of agent buffer.

This parameter can be greater than *DataLength* on invocation.

For channel message, send, and receive exits, any unused space on invocation can be used by the exit to expand the data in place. If this is done, the *DataLength* parameter must be set appropriately by the exit.

In the C programming language, this parameter is passed by address.

AgentBuffer (MQBYTE×AgentBufferLength) - input/output

Agent buffer.

The contents of this parameter depend upon the exit type:

- For a channel security exit, on invocation of the exit it contains a security message if *ExitReason* is MQXR_SEC_MSG. To send a security message back, the exit can either use this buffer or its own buffer (*ExitBufferAddr*).
- For a channel message exit, on invocation of the exit this parameter contains:
  - The transmission queue header (MQXQH), which includes the message descriptor (which itself contains the context information for the message), immediately followed by
  - The message data

  If the message is to proceed, the exit can do one of the following:
  - Leave the contents of the buffer untouched
  - Modify the contents in place (returning the new length of the data in *DataLength*; this must not be greater than *AgentBufferLength*)
  - Copy the contents to the *ExitBufferAddr*, making any required changes

  Any changes that the exit makes to the transmission queue header are not checked; however, erroneous modifications might mean that the message cannot be put at the destination.
- For a channel send or receive exit, on invocation of the exit this contains the transmission data. The exit can do one of the following:
  - Leave the contents of the buffer untouched
  - Modify the contents in place (returning the new length of the data in *DataLength*; this must not be greater than *AgentBufferLength*)
  - Copy the contents to the *ExitBufferAddr*, making any required changes

  The first 8 bytes of the data must not be changed by the exit.

ExitBufferLength (MQLONG) - input/output

Length of exit buffer.

On the first invocation of the exit, this parameter is set to zero. Thereafter whatever value is passed back by the exit, on each invocation, is presented to the exit next time it is invoked. The value is not used by the MCA.

Note: This parameter must not be used by exits written in programming languages which do not support the pointer data type.

ExitBufferAddr (MQPTR) - input/output

Address of exit buffer.

This parameter is a pointer to the address of a buffer of storage managed by the exit, where it can choose to return message or transmission data (depending upon the type of exit) to the agent if the buffer of the agent is or might not be large enough, or if it is more convenient for the exit to do so.

On the first invocation of the exit, the address passed to the exit is null. Thereafter whatever address is passed back by the exit, on each invocation, is presented to the exit the next time it is invoked.

Note: This parameter must not be used by exits written in programming languages that do not support the pointer data type.

C invocation

```
exitname (&ChannelExitParms, &ChannelDefinition,
         &DataLength, &AgentBufferLength, AgentBuffer,
         &ExitBufferLength, &ExitBufferAddr);
```

The parameters passed to the exit are declared as follows:

```
MQCXP   ChannelExitParms;   /* Channel exit parameter block */
MQCD    ChannelDefinition;  /* Channel definition */
MQLONG  DataLength;         /* Length of data */
MQLONG  AgentBufferLength;  /* Length of agent buffer */
```

```
MQBYTE  AgentBuffer[n];     /* Agent buffer */
MQLONG  ExitBufferLength;   /* Length of exit buffer */
MQPTR   ExitBufferAddr;     /* Address of exit buffer */
```

COBOL invocation

```
CALL 'exitname' USING CHANNELEXITPARMS, CHANNELDEFINITION,
                      DATALENGTH, AGENTBUFFERLENGTH, AGENTBUFFER,
                      EXITBUFFERLENGTH, EXITBUFFERADDR.
```

The parameters passed to the exit are declared as follows:

```
**   Channel exit parameter block
 01  CHANNELEXITPARMS.
     COPY CMQCXPV.
**   Channel definition
 01  CHANNELDEFINITION.
     COPY CMQCDV.
**   Length of data
 01  DATALENGTH         PIC S9(9) BINARY.
**   Length of agent buffer
 01  AGENTBUFFERLENGTH  PIC S9(9) BINARY.
**   Agent buffer
 01  AGENTBUFFER        PIC X(n).
**   Length of exit buffer
 01  EXITBUFFERLENGTH   PIC S9(9) BINARY.
**   Address of exit buffer
 01  EXITBUFFERADDR     POINTER.
```

RPG invocation (ILE)

```
C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
C                    CALLP     exitname(MQCXP : MQCD : DATLEN :
C                                       ABUFL : ABUF : EBUFL :
C                                       EBUF)
```

The prototype definition for the call is:

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
Dexitname         PR                  EXTPROC('exitname')
D* Channel exit parameter block
D MQCXP                         160A
D* Channel definition
D MQCD                         1328A
D* Length of data
D DATLEN                         10I 0
D* Length of agent buffer
D ABUFL                          10I 0
D* Agent buffer
D ABUF                             *   VALUE
D* Length of exit buffer
D EBUFL                          10I 0
D* Address of exit buffer
D EBUF                             *
```

System/390 assembler invocation

```
        CALL EXITNAME,(CHANNELEXITPARMS,CHANNELDEFINITION,DATALENGTH, X
              AGENTBUFFERLENGTH,AGENTBUFFER,EXITBUFFERLENGTH,         X
              EXITBUFFERADDR)
```

The parameters passed to the exit are declared as follows:

```
CHANNELEXITPARMS   CMQCXPA  ,      Channel exit parameter block
CHANNELDEFINITION  CMQCDA   ,      Channel definition
DATALENGTH         DS       F      Length of data
AGENTBUFFERLENGTH  DS       F      Length of agent buffer
```

```
AGENTBUFFER        DS       CL(n)  Agent buffer
EXITBUFFERLENGTH   DS       F      Length of exit buffer
EXITBUFFERADDR     DS       F      Address of exit buffer
```

Usage notes

1. The function performed by the channel exit is defined by the provider of the exit. The exit, however, must conform to the rules defined here and in the associated control block, the MQCXP.
2. The *ChannelDefinition* parameter passed to the channel exit might be one of several versions. See the *Version* field in the MQCD structure for more information.
3. If the channel exit receives an MQCD structure with the *Version* field set to a value greater than MQCD_VERSION_1, the exit must use the *ConnectionName* field in MQCD, in preference to the *ShortConnectionName* field.
4. In general, channel exits are allowed to change the length of message data. This can arise as a result of the exit adding data to the message, or removing data from the message, or compressing or encrypting the message. However, special restrictions apply if the message is a segment that contains only part of a logical message. In particular, there must be no net change in the length of the message as a result of the actions of complementary sending and receiving exits.

   For example, it is permissible for a sending exit to shorten the message by compressing it, but the complementary receiving exit must restore the original length of the message by decompressing it, so that there is no net change in the length of the message.

   This restriction arises because changing the length of a segment would cause the offsets of later segments in the message to be incorrect, and this would inhibit the ability of the queue manager to recognize that the segments formed a complete logical message.

MQ_CHANNEL_AUTO_DEF_EXIT - Channel auto-definition exit:

The MQ_CHANNEL_AUTO_DEF_EXIT call describes the parameters that are passed to the channel auto-definition exit called by the Message Channel Agent.

No entry point called MQ_CHANNEL_AUTO_DEF_EXIT is provided by the queue manager; the name MQ_CHANNEL_AUTO_DEF_EXIT is of no special significance because the names of the auto-definition exits are provided in the queue manager.

Syntax

MQ_CHANNEL_AUTO_DEF_EXIT *(ChannelExitParms, ChannelDefinition)*

Parameters

The MQ_CHANNEL_AUTO_DEF_EXIT call has the following parameters.

ChannelExitParms (MQCXP) - input/output

Channel exit parameter block.

This structure contains additional information relating to the invocation of the exit. The exit sets information in this structure to indicate how the MCA proceeds.

ChannelDefinition (MQCD) - input/output

Channel definition.

This structure contains parameters set by the administrator to control the behavior of channels which are created automatically. The exit sets information in this structure to modify the default behavior set by the administrator.

The MQCD fields listed must not be altered by the exit:

- *ChannelName*
- *ChannelType*
- *StrucLength*
- *Version*

If other fields are changed, the value set by the exit must be valid. If the value is not valid, an error message is written to the error log file or displayed on the console (as appropriate to the environment).

C invocation

```
exitname (&ChannelExitParms, &ChannelDefinition);
```

The parameters passed to the exit are declared as follows:

```
MQCXP  ChannelExitParms;   /* Channel exit parameter block */
MQCD   ChannelDefinition;  /* Channel definition */
```

COBOL invocation

```
     CALL 'exitname' USING CHANNELEXITPARMS, CHANNELDEFINITION.
```

The parameters passed to the exit are declared as follows:

```
**   Channel exit parameter block
 01  CHANNELEXITPARMS.
     COPY CMQCXPV.
**   Channel definition
 01  CHANNELDEFINITION.
     COPY CMQCDV.
```

RPG invocation (ILE)

```
     C*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     C                    CALLP     exitname(MQCXP : MQCD)
```

The prototype definition for the call is:

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     Dexitname         PR                  EXTPROC('exitname')
     D* Channel exit parameter block
     D MQCXP                          160A
     D* Channel definition
     D MQCD                          1328A
```

System/390 assembler invocation

```
         CALL EXITNAME,(CHANNELEXITPARMS,CHANNELDEFINITION)
```

The parameters passed to the exit are declared as follows:

```
CHANNELEXITPARMS   CMQCXPA  ,  Channel exit parameter block
CHANNELDEFINITION  CMQCDA   ,  Channel definition
```

Usage notes

1. The function performed by the channel exit is defined by the provider of the exit. The exit, however, must conform to the rules defined here and in the associated control block, the MQCXP.
2. The *ChannelExitParms* parameter passed to the channel auto-definition exit is an MQCXP structure. The version of MQCXP passed depends on the environment in which the exit is running; see the description of the *Version* field in "MQCXP - Channel exit parameter" on page 2394 for details.
3. The *ChannelDefinition* parameter passed to the channel auto-definition exit is an MQCD structure. The version of MQCD passed depends on the environment in which the exit is running; see the description of the *Version* field in "MQCD - Channel definition" on page 2348 for details.

MQXWAIT - Wait in exit:

The MQXWAIT call waits for an event to occur. It can be used only from a channel exit on z/OS.

The use of MQXWAIT helps to avoid performance problems that might otherwise occur if a channel exit does something that causes a wait. The event MQXWAIT is waiting on is signaled by an MVS ECB (event control block). The ECB is described in the MQXWD control block description.

Syntax

MQXWAIT *(Hconn, WaitDesc, CompCode, Reason)*

Parameters

The MQXWAIT call has the following parameters.

Hconn (MQHCONN) - input

Connection handle.

This handle represents the connection to the queue manager. The value of *Hconn* was returned by a previous MQCONN call issued in the same or earlier invocation of the exit.

WaitDesc (MQXWD) - input/output

Wait descriptor.

This parameter describes the event to wait for. See "MQXWD - Exit wait descriptor" on page 2411 for details of the fields in this structure.

CompCode (MQLONG) - output

Completion code.

It is one of the following codes:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

MQRC_ADAPTER_NOT_AVAILABLE
(2204, X'89C') Adapter not available.

MQRC_OPTIONS_ERROR
(2046, X'7FE') Options not valid or not consistent.

MQRC_XWAIT_CANCELED
(2107, X'83B') MQXWAIT call canceled.

MQRC_XWAIT_ERROR
(2108, X'83C') Invocation of MQXWAIT call not valid.

C invocation

```
MQXWAIT (Hconn, &WaitDesc, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQXWD    WaitDesc;  /* Wait descriptor */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

System/390 assembler invocation

```
        CALL MQXWAIT,(HCONN,WAITDESC,COMPCODE,REASON)
```

Declare the parameters as follows:

```
HCONN     DS       F  Connection handle
WAITDESC  CMQXWDA  ,  Wait descriptor
COMPCODE  DS       F  Completion code
REASON    DS       F  Reason code qualifying COMPCODE
```

MQCD - Channel definition:

The MQCD structure contains the parameters which control execution of a channel. It is passed to each channel exit that is called from a Message Channel Agent (MCA).

For more information about channel exits, see "MQ_CHANNEL_EXIT - Channel exit" on page 2341. The description in this topic relates both to message channels and to MQI channels.

Exit name fields

When an exit is called, the relevant field from *SecurityExit*, *MsgExit*, *SendExit*, *ReceiveExit*, and *MsgRetryExit* contains the name of the exit currently being invoked. The meaning of the name in these fields depends on the environment in which the MCA is running. Except where noted, the name is left-aligned within the field, with no embedded blanks; the name is padded with blanks to the length of the field. In the descriptions that follow, square brackets ([ ]) denote optional information:

UNIX systems

The exit name is the name of a dynamically loadable module or library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path:

[path]library(function)

The name is limited to a maximum of 128 characters.

z/OS The exit name is the name of a load module that is valid for specification on the EP parameter of the LINK or LOAD macro. The name is limited to a maximum of eight characters.

Windows

The exit name is the name of a dynamic-link library, suffixed with the name of a function residing in that library. The function name must be enclosed in parentheses. The library name can optionally be prefixed with a directory path and drive:

[d:][path]library(function)

The name is limited to a maximum of 128 characters.

IBM i The exit name is a 10 byte program name followed by a 10 byte library name. If the names are less than 10 bytes long, each name is padded with blanks to make it 10 bytes. The library name can be **LIBL* except when calling a channel auto-definition exit, in which case a fully qualified name is required.

Changing MQCD fields in a channel exit

A channel exit can change fields in the MQCD. The changed value remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance. The changed MQCD is also used by the MCA for its normal processing during the continuing lifetime of the channel.

The following MQCD fields must not be altered by the exit:

- ChannelName
- ChannelType
- StrucLength
- Version

Related reference:

"Fields"
This topic lists all the fields in the MQCD structure and describes each field.

"C declaration" on page 2380
This declaration is the C declaration for the MQCD structure.

"COBOL declaration" on page 2382
This declaration is the COBOL declaration for the MQCD structure.

"RPG declaration (ILE)" on page 2385
This declaration is the RPG declaration for the MQCD structure.

"System/390 assembler declaration" on page 2388
This declaration is the System/390 assembler declaration for the MQCD structure.

"Visual Basic declaration" on page 2390
This declaration is the Visual Basic declaration of the MQCD structure.

"Changing MQCD fields in a channel exit" on page 2392
A channel exit can change fields in the MQCD. However, these changes are not typically acted on, except in the circumstances listed.

Fields:

This topic lists all the fields in the MQCD structure and describes each field.

BatchHeartbeat (MQLONG):

This field specifies the time interval that is used to trigger a batch heartbeat for the channel.

Batch heartbeating allows sender channels to determine whether the remote channel instance is still active before going indoubt. A batch heartbeat occurs if a sender channel has not communicated with the remote channel instance within the specified time interval.

The value is in the range 0 through 999 999; the units are milliseconds. A value of zero indicates that batch heartbeating is not enabled.

This field is relevant only for channels that have a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

BatchInterval (MQLONG):

This field specifies the approximate time in milliseconds that a channel keeps a batch open, if fewer than *BatchSize* messages have been transmitted in the current batch.

If *BatchInterval* is greater than zero, the batch is terminated by whichever of the following events occur first:

- *BatchSize* messages have been sent, or
- *BatchInterval* milliseconds have elapsed since the start of the batch.

If *BatchInterval* is zero, the batch is terminated by whichever of the following events occur first:

- *BatchSize* messages have been sent, or
- the transmission queue becomes empty.

BatchInterval must be in the range zero through 999 999 999.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present when *Version* is less than MQCD_VERSION_4.

BatchSize (MQLONG):

This field specifies the maximum number of messages that can be sent through a channel before synchronizing the channel.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

ChannelMonitoring (MQLONG):

This field specifies the current level of monitoring data collection for the channel.

This field is not relevant for channels with a ChannelType of MQCHT_CLNTCONN.

It is one of the following values:

- MQMON_OFF
- MQMON_LOW
- MQMON_MEDIUM
- MQMON_HIGH

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_8.

ChannelName (MQCHAR20):

This field specifies the channel definition name.

There must be a channel definition of the same name at the remote machine to be able to communicate.

The name must use only the characters:

- Uppercase A-Z
- Lowercase a-z
- Numerics 0-9

- Period (.)
- Forward slash (/)
- Underscore (_)
- Percent sign (%)

and be padded to the right with blanks. Leading or embedded blanks are not allowed.

The length of this field is given by MQ_CHANNEL_NAME_LENGTH.

ChannelStatistics (MQLONG):

This field specifies the current level of statistics data collection for the channel.

This field is not relevant for channels with a ChannelType of MQCHT_CLNTCONN.

It is one of the following values:
- MQMON_OFF
- MQMON_LOW
- MQMON_MEDIUM
- MQMON_HIGH

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_8.

ChannelType (MQLONG):

This field specifies the type of channel.

It is one of the following values:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_CLNTCONN
Client connection.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLUSSDR
Cluster sender.

MQCHT_CLUSRCVR
Cluster receiver.

ClientChannelWeight (MQLONG):

This field specifies a weighting to influence which client-connection channel definition is used.

The ClientChannelWeight attribute is used so that client channel definitions can be selected at random based on their weighting when more than one suitable definition is available. When a client issues an MQCONN requesting connection to a queue manager group, by specifying a queue manager name starting with an asterisk, and more than one suitable channel definition is available in the client channel definition table (CCDT), the definition to use is randomly selected based on the weighting, with any applicable ClientChannelWeight(0) definitions selected first in alphabetical order.

Specify a value in the range 0 - 99. The default is 0.

A value of 0 indicates that no load balancing is performed and applicable definitions are selected in alphabetical order. To enable load balancing choose a value in the range 1 - 99 where 1 is the lowest weighting and 99 is the highest. The distribution of messages between two or more channels with non-zero weightings is proportional to the ratio of those weightings. For example, three channels with ClientChannelWeight values of 2, 4, and 14 are selected approximately 10%, 20%, and 70% of the time. This distribution is not guaranteed.

This attribute is valid for the client-connection channel type only.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_9.

ClusterPtr (MQPTR):

This field specifies the address a list of cluster names.

If *ClustersDefined* is greater than zero, this address is the address of a list of cluster names. The channel belongs to each cluster listed.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

ClustersDefined (MQLONG):

This field specifies the number of clusters to which the channel belongs.

This field is the number of cluster names pointed to by *ClusterPtr*. It is zero or greater.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

CLWLChannelPriority (MQLONG):

This field specifies the cluster workload channel priority.

The workload manager choose algorithm selects a destination with the highest priority from the set of destinations selected based on rank. If there are two possible destination queue managers, this attribute can be used to make one queue manager failover onto the other queue manager. All the messages go to the queue manager with the highest priority until that ends, then the messages go to the queue manager with the next highest priority.

The value is in the range 0 through 9. The default is 0.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see Configuring a queue manager cluster.

CLWLChannelRank (MQLONG):

This field specifies the cluster workload channel rank.

The workload manager choose algorithm selects a destination with the highest rank. When the final destination is a queue manager on a different cluster, you can set the rank of intermediate gateway queue managers (at the intersection of neighboring clusters) so the choose algorithm correctly chooses a destination queue manager nearer the final destination.

The value is in the range 0 through 9. The default is 0.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see Configuring a queue manager cluster.

CLWLChannelWeight (MQLONG):

This field specifies the cluster workload channel weight.

Cluster workload channel weight.

The workload manager choose algorithm uses the "weight" attribute of the channel to the skew the destination choice so that more messages can be sent to a particular machine. For example, you can give a channel on a large UNIX server a larger "weight" than another channel on small desktop PC, and the choose algorithm chooses the UNIX server more frequently than the PC.

The value is in the range 1 through 99. The default is 50.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

For further information, see Configuring a queue manager cluster.

ConnectionAffinity (MQLONG):

This field specifies whether client applications that connect multiple times using the same queue manager name, use the same client channel.

Use this attribute when multiple applicable channel definitions are available.

The value is one of the following:

MQCAFTY_PREFERRED
The first connection in a process reading a client channel definition table (CCDT) creates a list of applicable definitions based on the weighting with any applicable CLNTWGHT(0) definitions first and in alphabetical order. Each connection in the process attempts to connect using the first definition in the list. If a connection is unsuccessful the next definition is used. Unsuccessful definitions with CLNTWGHT values other than 0 are moved to the end of the list. CLNTWGHT(0) definitions remain at the start of the list and are selected first for each connection.

Each client process with the same host name always creates the same list.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET) the list is updated if the CCDT has been modified since the list was created.

This value is the default value.

MQCAFTY_NONE
The first connection in a process reading a CCDT creates a list of applicable definitions. All connections in a process select an applicable definition based on the weighting with any applicable CLNTWGHT(0) definitions selected first in alphabetical order.

For client applications written in C, C++, or the .NET programming framework (including fully managed .NET) the list is updated if the CCDT has been modified since the list was created.

This attribute is valid for the client-connection channel type only.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_9.

ConnectionName (MQCHAR264):

This field specifies the connection name for the channel.

For cluster-receiver channels (when specified) CONNAME relates to the local queue manager, and for other channels it relates to the target queue manager. The value you specify depends on the transmission protocol (*TransportType*) to be used:

- For MQXPT_LU62, it is the fully-qualified name of the partner Logical Unit.
- For MQXPT_NETBIOS, it is the NetBIOS name defined on the remote machine.
- For MQXPT_TCP, it is either the host name, the network address of the remote machine specified in IPv4 dotted decimal or IPv6 hexadecimal format, or the local machine for cluster-receiver channels.
- For MQXPT_SPX, it is an SPX-style address comprising a 4 byte network address, a 6 byte node address, and a 2 byte socket number.

When defining a channel, this field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_RECEIVER. However, when the channel definition is passed to an exit, this field contains the address of the partner, whatever the channel type.

The length of this field is given by MQ_CONN_NAME_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

DataConversion (MQLONG):

This field specifies whether the sending message channel agent attempts conversion of the application message data if the receiving message channel agent is unable to perform this conversion.

This field applies only to messages that are not segments of logical messages; the MCA never attempts to convert messages which are segments.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR. It is one of the following:

MQCDC_SENDER_CONVERSION
Conversion by sender.

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

DefReconnect (MQLONG):

The DefReconnect channel attribute sets the default reconnection attribute value for a client connection channel.

The default automatic client reconnection option. You can configure a IBM WebSphere MQ MQI client to automatically reconnect a client application. The IBM WebSphere MQ MQI client tries to reconnect to a queue manager after a connection failure. It tries to reconnect without the application client issuing an MQCONN or MQCONNX MQI call.

Reconnection is an MQCONNX option. By using the DefReconnect channel attribute you can add reconnection behavior to existing applications that use MQCONN. You can also change the reconnection behavior of applications that use MQCONNX.

You can also set the DefRecon value from the `mqclient.ini` file to set or modify reconnection behavior. The DefRecon value from the `mqclient.ini` file takes precedence over the DefReconnect channel attribute.

Syntax

DefReconnect(**MQRCN_NO** | **MQRCN_YES** | **MQRCN_Q_MGR** | **MQRCN_DISABLED**)

Parameters

MQRCN_NO
MQRCN_NO is the default value.

Unless overridden by MQCONNX, the client is not reconnected automatically.

MQRCN_YES
Unless overridden by MQCONNX, the client reconnects automatically.

MQRCN_Q_MGR
Unless overridden by MQCONNX, the client reconnects automatically, but only to the same queue manager. The `QMGR` option has the same effect as `MQCNO_RECONNECT_Q_MGR`.

MQRCN_DISABLED
Reconnection is disabled, even if requested by the client program using the MQCONNX MQI call.

Automatic client reconnection is not supported by IBM WebSphere MQ classes for Java.

Table 238. Automatic reconnection depends on the values set in the application and in the channel definition

DefReconnect	**Reconnection options set in the application**			
	`MQCNO_RECONNECT`	`MQCNO_RECONNECT_Q_MGR`	`MQCNO_RECONNECT_AS_DEF`	`MQCNO_RECONNECT_DISABLED`
MQRCN_NO	YES	QMGR	NO	NO
MQRCN_YES	YES	QMGR	YES	NO
MQRCN_Q_MGR	YES	QMGR	QMGR	NO
MQRCN_DISABLED	NO	NO	NO	NO

Related reference:

Connection options
Options that control the action of MQCONNX.

Related information:

Automatic client reconnection

Channel and client reconnection

CHANNELS stanza of the client configuration file

Desc (MQCHAR64):

This field can be used for descriptive commentary.

The content of the field is of no significance to Message Channel Agents. However, it must contain only characters that can be displayed. It cannot contain any null characters; if necessary, it is padded to the right with blanks. In a DBCS installation, the field can contain DBCS characters (subject to a maximum field length of 64 bytes).

Note: If this field contains characters that are not in the character set of the queue manager (as defined by the *CodedCharSetId* queue manager attribute), those characters might be translated incorrectly if this field is sent to another queue manager.

The length of this field is given by MQ_CHANNEL_DESC_LENGTH.

DiscInterval (MQLONG):

This field specifies the maximum time in seconds for which the channel waits for a message to arrive on the transmission queue, before terminating the channel.

In other words, it specifies the disconnect interval.

The A value of zero causes the MCA to wait indefinitely.

For server-connection channels using the TCP protocol, the interval represents the client inactivity disconnect value, specified in seconds. If a server-connection has received no communication from its partner client for this duration, it terminates the connection. The server-connection inactivity interval only applies between WebSphere MQ API calls from a client, so no client is disconnected during a long-running MQGET with wait call.

This attribute is not applicable for server-connection channels using protocols other than TCP.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, MQCHT_CLUSRCVR, or MQCHT_SVRCONN.

ExitDataLength (MQLONG):

This field specifies length in bytes of each of the user data items in the lists of exit user data items addressed by the *MsgUserDataPtr*, *SendUserDataPtr*, and *ReceiveUserDataPtr* fields.

This length is not necessarily the same as MQ_EXIT_DATA_LENGTH.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ExitNameLength (MQLONG):

This field specifies the length in bytes of each of the names in the lists of exit names addressed by the *MsgExitPtr*, *SendExitPtr*, and *ReceiveExitPtr* fields.

This length is not necessarily the same as MQ_EXIT_NAME_LENGTH.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

HdrCompList [2] (MQLONG):

This field specifies the list of header data compression techniques which are supported by the channel.

The list contains one or more of the following values:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

Unused values in the array are set to MQCOMPRESS_NOT_AVAILABLE.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

HeartbeatInterval (MQLONG):

This field specifies the time in seconds between heartbeat flows.

The interpretation of this field depends on the channel type, as follows:

- For a channel type of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR, this field is the time in seconds between heartbeat flows passed from the sending MCA when there are no messages on the transmission queue. This gives the receiving MCA the opportunity to quiesce the channel. To be useful, *HeartbeatInterval* must be less than *DiscInterval*.
- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN with the MQCD Sharing Conversations field set to zero, this field is the time in seconds between heartbeat flows passed from the server MCA when that MCA has issued an MQGET call with the MQGMO_WAIT option on behalf of a client application. This allows the server MCA to handle situations where the client connection fails during an MQGET with MQGMO_WAIT.
- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN with the MQCD Sharing Conversations field set to a non-zero value, this field is the time in seconds between heartbeat flow when there are no data flows sent or received. This allows the channel to be quiesced efficiently.

The value is in the range 0 through 999 999. The value that is used is the larger of the values specified at the sending side and receiving side unless a value of 0 is specified at either side, in which case no heartbeat exchange occurs.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

KeepAliveInterval (MQLONG):

This field specifies the value passed to the communications stack for keepalive timing for the channel.

The value is applicable for the TCP/IP and SPX communications protocols, though not all implementations support this parameter.

The value is in the range 0 through 99 999; the units are seconds. A value of zero indicates that channel keepalive is not enabled, although keepalive might still occur if TCP/IP keepalive (rather than channel keepalive) is enabled. The following special value is also valid:

MQKAI_AUTO
Automatic.

This value indicates that the keepalive interval is calculated from the negotiated heartbeat interval, as follows:

- If the negotiated heartbeat interval is greater than zero, the keepalive interval that is used is the heartbeat interval plus 60 seconds.
- If the negotiated heartbeat interval is zero, the keepalive interval that is used is zero.

- On z/OS, TCP/IP keepalive occurs when TCPKEEP(YES) is specified on the queue manager object.
- In other environments, TCP/IP keepalive occurs when the `KEEPALIVE=YES` parameter is specified in the TCP stanza in the distributed queuing configuration file.

This field is relevant only for channels that have a *TransportType* of MQXPT_TCP or MQXPT_SPX.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

LocalAddress (MQCHAR48):

This field specifies the local TCP/IP address defined for the channel for outbound communications.

This field is blank if no specific address is defined for outbound communications. The address can optionally include a port number or range of port numbers. The format of this address is:

```
[ip-addr][(low-port[,high-port])]
```

where square brackets ([ ]) denote optional information, `ip-addr` is specified in IPv4 dotted decimal, IPv6 hexadecimal, or alphanumeric form, and `low-port` and `high-port` are port numbers enclosed in parentheses. All are optional.

A specific IP address, port, or port range for outbound communications is useful in recovery scenarios where a channel is restarted on a different TCP/IP stack.

LocalAddress is similar in form to *ConnectionName*, but must not be confused with it. *LocalAddress* specifies the characteristics of the local communications, whereas *ConnectionName* specifies how to reach a remote queue manager.

This field is relevant only for channels with a *TransportType* of MQXPT_TCP, and a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_LOCAL_ADDRESS_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_7.

LongMCAUserIdLength (MQLONG):

This field specifies the length in bytes of the full MCA user identifier pointed to by *LongMCAUserIdPtr*.

This field is not relevant for channels with a *ChannelType* of MQCHT_CLNTCONN.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongMCAUserIdPtr (MQPTR):

This field specifies the address of the long MCA user identifier.

If *LongMCAUserIdLength* is greater than zero, this field is the address of the full MCA user identifier. The length of the full identifier is given by *LongMCAUserIdLength*. The first 12 bytes of the MCA user identifier are also contained in the field *MCAUserIdentifier*.

See the description of the *MCAUserIdentifier* field for details of the MCA user identifier.

This field is not relevant for channels with a *ChannelType* of MQCHT_SDR, MQCHT_SVR, MQCHT_CLNTCONN, or MQCHT_CLUSSDR.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongRemoteUserIdLength (MQLONG):

This field specifies the length in bytes of the full remote user identifier pointed to by *LongRemoteUserIdPtr*.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongRemoteUserIdPtr (MQPTR):

This field specifies the address of the long remote user identifier.

If *LongRemoteUserIdLength* is greater than zero, this flag is the address of the full remote user identifier. The length of the full identifier is given by *LongRemoteUserIdLength*. The first 12 bytes of the remote user identifier are also contained in the field *RemoteUserIdentifier*.

See the description of the *RemoteUserIdentifier* field for details of the remote user identifier.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_6.

LongRetryCount (MQLONG):

This field specifies the count used after the count specified by the *ShortRetryCount* has been exhausted.

It specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by *LongRetryInterval*, before logging an error to the operator.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

LongRetryInterval (MQLONG):

This field specifies the maximum number of seconds to wait before reattempting connection to the remote machine.

The interval between retries can be extended if the channel has to wait to become active.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

MaxInstances (MQLONG):

This field specifies the maximum number of simultaneous instances of an individual server-connection channel that can be started.

This field is used only on server-connection channels.

The field can have a value in the range 0 - 999 999 999. A value of zero prevents all client access.

The default value of this field is 999 999 999.

If the value of this field is reduced to a number that is less than the number of instances of the server-connection channel that are currently running, then those running instances are not affected. However, new instances cannot start until sufficient existing instances have ceased to run so that the number of currently running instances is less than the value of the field.

MaxInstancesPerClient (MQLONG):

This field specifies the maximum number of simultaneous instances of an individual server-connection channel that can be started from a single client.

In this context, connections that originate from the same remote network address are regarded as coming from the same client.

This field is used only on server-connection channels.

The field can have a value in the range 0 - 999 999 999. A value of zero prevents all client access.

The default value of this field is 999 999 999.

If the value of this field is reduced to a number that is less than the number of instances of the server-connection channel that are currently running from individual clients, then those running instances are not affected. However, new instances from any of those clients cannot start until sufficient existing instances have ceased to run such that the number of currently running instances, originating from the client attempting to start a new one, is less than the value of the field.

MaxMsgLength (MQLONG):

This field specifies the maximum message length that can be transmitted on the channel.

This is compared with the value for the remote channel and the actual maximum is the lower of the two values.

MCAName (MQCHAR20):

This field is a reserved field.

The value of this field is blank.

The length of this field is given by MQ_MCA_NAME_LENGTH.

MCASecurityId (MQBYTE40):

This field specifies the security identifier for the MCA.

This field is not relevant for channels with a *ChannelType* of MQCHT_CLNTCONN.

The following special value indicates that there is no security identifier:

MQSID_NONE
No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQSID_NONE_ARRAY is also defined; this constant has the same value as MQSID_NONE, but is an array of characters instead of a string.

This is an input/output field to the exit. The length of this field is given by MQ_SECURITY_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_6.

MCAType (MQLONG):

This field specifies the type of message channel agent program.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value is one of the following:

MQMCAT_PROCESS
Process.

The message channel agent runs as a separate process.

MQMCAT_THREAD
Thread (IBM i, UNIX, and Windows).

The message channel agent runs as a separate thread.

This field is not present when *Version* is less than MQCD_VERSION_2.

MCAUserIdentifier (MQCHAR12):

This field specifies the user identifier for the message channel agent (MCA).

This field uses the first 12 bytes of the MCA user identifier, and can be set by a security agent.

There are two fields that contain the MCA user identifier:

- *MCAUserIdentifier* contains the first 12 bytes of the MCA user identifier, and is padded with blanks if the identifier is shorter than 12 bytes. *MCAUserIdentifier* can be blank.
- *LongMCAUserIdPtr* points to the full MCA user identifier, which can be longer than 12 bytes. Its length is given by *LongMCAUserIdLength*. The full identifier contains no trailing blanks, and is not null-terminated. If the identifier is blank, *LongMCAUserIdLength* is zero, and the value of *LongMCAUserIdPtr* is undefined.

  Note: *LongMCAUserIdPtr* is not present if *Version* is less than MQCD_VERSION_6.

If the MCA user identifier is nonblank, it specifies the user identifier to be used by the message channel agent for authorization to access WebSphere MQ resources. For channel types MQCHT_REQUESTER, MQCHT_RECEIVER, and MQCHT_CLUSRCVR, if PutAuthority is MQPA_DEFAULT this is the user identifier used for authorization checks for the put operation to destination queues.

If the MCA user identifier is blank, the message channel agent uses its default user identifier.

The MCA user identifier can be set by a security exit to indicate the user identifier that the message channel agent must use. The exit can change either *MCAUserIdentifier*, or the string pointed at by *LongMCAUserIdPtr*. If both are changed but differ from each other, the MCA uses *LongMCAUserIdPtr* in preference to *MCAUserIdentifier*. If the exit changes the length of the string addressed by *LongMCAUserIdPtr*, *LongMCAUserIdLength* must be set correspondingly. If the exit increases the length of the identifier, the exit must allocate storage of the required length, set that storage to the required identifier, and place the address of that storage in *LongMCAUserIdPtr*. The exit is responsible for freeing that storage when the exit is later invoked with the MQXR_TERM reason.

For channels with a *ChannelType* of MQCHT_SVRCONN, if *MCAUserIdentifier* in the channel definition is blank, any user identifier transferred from the client is copied into it. This user identifier (after any modification by the security exit at the server) is the one which the client application is assumed to be running under.

The MCA user identifier is not relevant for channels with a *ChannelType* of MQCHT_SDR, MQCHT_SVR, MQCHT_CLNTCONN, MQCHT_CLUSSDR.

This is an input/output field to the exit. The length of this field is given by MQ_USER_ID_LENGTH. This field is not present when *Version* is less than MQCD_VERSION_2.

ModeName (MQCHAR8):

This field specifies the LU 6.2 mode name.

This field is relevant only if the transmission protocol (*TransportType*) is MQXPT_LU62, and the *ChannelType* is not MQCHT_SVRCONN or MQCHT_RECEIVER.

This field is always blank. The information is contained in the communications Side Object instead.

The length of this field is given by MQ_MODE_NAME_LENGTH.

MsgCompList [16] (MQLONG):

This field specifies the list of message data compression techniques which are supported by the channel.

The list contains one or more of the following values:

MQCOMPRESS_NONE
No message data compression is performed.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

Unused values in the array are set to MQCOMPRESS_NOT_AVAILABLE.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_8.

MsgExit (MQCHARn):

This field specifies the channel message exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after a message has been retrieved from the transmission queue (sender or server), or immediately before a message is put to a destination queue (receiver or requester).

  The exit is given the entire application message and transmission queue header for modification.
- At initialization and termination of the channel.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN; a message exit is never invoked for such channels.

See "MQCD - Channel definition" on page 2348 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

MsgExitPtr (MQPTR):

This field specifies the address of the first *MsgExit* field.

If *MsgExitsDefined* is greater than zero, this address is the address of the list of names of each channel message exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another - one for each exit.

Any changes made to these names by an exit are preserved, although the message channel exit takes no explicit action - it does not change which exits are invoked.

If *MsgExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

MsgExitsDefined (MQLONG):

This field specifies the number of channel message exits defined in the chain.

It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

MsgRetryCount (MQLONG):

This field specifies the number of times MCA tries to put the message, after the first attempt has failed.

This field indicates the number of times that the MCA tries the open or put operation, if the first MQOPEN or MQPUT fails with completion code MQCC_FAILED. The effect of this attribute depends on whether *MsgRetryExit* is blank or nonblank:

- If *MsgRetryExit* is blank, the *MsgRetryCount* attribute controls whether the MCA attempts retries. If the attribute value is zero, no retries are attempted. If the attribute value is greater than zero, the retries are attempted at intervals given by the *MsgRetryInterval* attribute.

  Retries are attempted only for the following reason codes:
  - MQRC_PAGESET_FULL
  - MQRC_PUT_INHIBITED
  - MQRC_Q_FULL

  For other reason codes, the MCA proceeds immediately to its normal failure processing, without retrying the failing message.
- If *MsgRetryExit* is nonblank, the *MsgRetryCount* attribute does not affect the MCA; instead it is the message-retry exit which determines how many times the retry is attempted, and at what intervals; the exit is invoked even if the *MsgRetryCount* attribute is zero.

  The *MsgRetryCount* attribute is made available to the exit in the MQCD structure, but the exit it not required to honor it - retries continue indefinitely until the exit returns MQXCC_SUPPRESS_FUNCTION in the *ExitResponse* field of MQCXP.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

This field is not present when *Version* is less than MQCD_VERSION_3.

MsgRetryExit (MQCHARn):

This field specifies the channel message retry exit name.

The message retry exit is an exit that is invoked by the MCA when the MCA receives a completion code of MQCC_FAILED from an MQOPEN or MQPUT call. The purpose of the exit is to specify a time interval for which the MCA waits before trying the MQOPEN or MQPUT operation again. Alternatively, the exit can be set to not try the operation again.

The exit is invoked for all reason codes that have a completion code of MQCC_FAILED - the settings of the exit determine which reason codes it wants the MCA to try again, for how many attempts, and at what time intervals.

When the operation is not to be attempted any more, the MCA performs its normal failure processing; this processing includes generating an exception report message (if specified by the sender), and either placing the original message on the dead-letter queue or discarding the message (according to whether the sender specified MQRO_DEAD_LETTER_Q or MQRO_DISCARD_MSG). Failures involving the dead-letter queue (for example, dead-letter queue full) do not cause the message-retry exit to be invoked.

If the exit name is nonblank, the exit is called at the following times:

- Immediately before performing the wait before trying to deliver a message again
- At initialization and termination of the channel

See "MQCD - Channel definition" on page 2348 for a description of the content of this field in various environments.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

This field is not present when *Version* is less than MQCD_VERSION_3.

MsgRetryInterval (MQLONG):

This field specifies the minimum interval in milliseconds after which the open or put operation is retried.

The effect of this attribute depends on whether *MsgRetryExit* is blank or nonblank:

- If *MsgRetryExit* is blank, the *MsgRetryInterval* attribute specifies the minimum period that the MCA waits before retrying a message, if the first MQOPEN or MQPUT fails with completion code MQCC_FAILED. A value of zero means that the retry will be performed as soon as possible after the previous attempt. Retries are performed only if *MsgRetryCount* is greater than zero.

  This attribute is also used as the wait time if the message-retry exit returns an invalid value in the *MsgRetryInterval* field in MQCXP.
- If *MsgRetryExit* is not blank, the *MsgRetryInterval* attribute does not affect the MCA; instead it is the message-retry exit which determines how long the MCA waits. The *MsgRetryInterval* attribute is made available to the exit in the MQCD structure, but the exit it not required to honor it.

The value is in the range 0 through 999 999 999.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

This field is not present when *Version* is less than MQCD_VERSION_3.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_4.

MsgRetryUserData (MQCHAR32):

This field specifies the channel message retry exit user data.

This data is passed to the channel message-retry exit in the *ExitData* field of the *ChannelExitParms* parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. Such changes do not affect the channel definition used by other MCA instances. Any characters (including binary data) can be used.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR.

The length of this field is given by MQ_EXIT_DATA_LENGTH. This field is not present when *Version* is less than MQCD_VERSION_3.

This field is not relevant in WebSphere MQ for IBM i.

MsgUserData (MQCHAR32):

This field specifies channel message exit user data.

This data is passed to the channel message exit in the *ExitData* field of the *ChannelExitParms* parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. Such changes do not affect the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This field is not relevant in WebSphere MQ for IBM i.

MsgUserDataPtr (MQPTR):

This field specifies the address of the first *MsgUserData* field.

If *MsgExitsDefined* is greater than zero, this address is the address of the list of user data items for each channel message exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another - one for each exit. If the number of user data items defined is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *MsgExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

NetworkPriority (MQLONG):

This field specifies the priority of the network connection for the channel.

When multiple paths to a particular destination are available, the path with the highest priority is chosen. The value is in the range 0 through 9; 0 is the lowest priority.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLUSSDR or MQCHT_CLUSRCVR.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_5.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_6.

NonPersistentMsgSpeed (MQLONG):

This field specifies the speed at which nonpersistent messages travel through the channel.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value is one of the following:

MQNPMS_NORMAL
Normal speed.

If a channel is defined to be MQNPMS_NORMAL, nonpersistent messages travel through the channel at normal speed. This has the advantage that these messages are not lost if there is a channel failure. Also, persistent and nonpersistent messages on the same transmission queue maintain their order relative to each other.

MQNPMS_FAST
Fast speed.

If a channel is defined to be MQNPMS_FAST, nonpersistent messages travel through the channel at fast speed. This improves the throughput of the channel, but means that nonpersistent messages are lost if there is a channel failure. Also, it is possible for nonpersistent messages to jump ahead of persistent messages waiting on the same transmission queue, that is, the order of nonpersistent messages is not maintained relative to persistent messages. However the order of nonpersistent messages relative to each other is maintained. Similarly, the order of persistent messages relative to each other is maintained.

Password (MQCHAR12):

This field specifies the password used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This field can be nonblank only on UNIX systems, and Windows, and is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, or MQCHT_CLNTCONN. On z/OS, this field is not relevant.

The length of this field is given by MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

This field is not present if *Version* is less than MQCD_VERSION_2.

PropertyControl (MQLONG):

This field specifies what happens to properties of messages when the message is about to be sent to a V6 or prior queue manager (a queue manager that does not understand the concept of a property descriptor).

The value can be:

MQPROP_COMPATIBILITY
If the message contains a property with a prefix of **mcd.**, **jms.**, **usr.**, or **mqext.**, all message properties are delivered to the application in an MQRFH2 header. Otherwise all properties of the message, except those properties contained in the message descriptor (or extension), are discarded and are no longer accessible to the application.

This value is the default value; it allows applications, which expect JMS-related properties to be in an MQRFH2 header in the message data, to continue to work unmodified.

MQPROP_NONE
All properties of the message, except those properties in the message descriptor (or extension), are removed from the message before the message is sent to the remote queue manager.

MQPROP_ALL
All properties of the message are included with the message when it is sent to the remote queue manager. The properties, except those properties in the message descriptor (or extension), are placed in one or more MQRFH2 headers in the message data.

This attribute is applicable to Sender, Server, Cluster Sender, and Cluster Receiver channels.

"MQIA_* (Integer Attribute Selectors)" on page 1442

"MQPROP_* (Queue and Channel Property Control Values and Maximum Properties Length)" on page 1474

PutAuthority (MQLONG):

This field specifies whether the user identifier in the context information associated with a message is used to establish authority to put the message to the destination queue.

This field is relevant only for channels with a *ChannelType* of MQCHT_REQUESTER, MQCHT_RECEIVER, or MQCHT_CLUSRCVR. It is one of the following:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

MQPA_ALTERNATE_OR_MCA
The user ID from the UserIdentifier field of the message descriptor is used. Any user ID received from the network is not used. This value is supported only on z/OS.

MQPA_ONLY_MCA
The default user ID is used. Any user ID received from the network is not used. This value is supported only on z/OS.

QMgrName (MQCHAR48):

This field specifies the name of the queue manager that an exit can connect to.

For channels with a *ChannelType* other than MQCHT_CLNTCONN, this field is the name of the queue manager that an exit can connect to, which on UNIX, Linux and Windows systems, is always nonblank.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

ReceiveExit (MQCHARn):

This field specifies the channel receive exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before the received network data is processed.

  The exit is given the complete transmission buffer as received. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

See "MQCD - Channel definition" on page 2348 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

ReceiveExitPtr (MQPTR):

This field specifies the address of the first *ReceiveExit* field.

If *ReceiveExitsDefined* is greater than zero, this address is the address of the list of names of each channel receive exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *ReceiveExitsDefined* fields adjoining one another - one for each exit.

Any changes made to these names by an exit are preserved, although the message channel exit takes no explicit action - it does not change which exits are invoked.

If *ReceiveExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ReceiveExitsDefined (MQLONG):

This field specifies the number of channel receive exits defined in the chain.

It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

ReceiveUserData (MQCHAR32):

This channel specifies channel receive exit user data.

This data is passed to the channel receive exit in the *ExitData* field of the *ChannelExitParms* parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes do not affect the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This field is not relevant in WebSphere MQ for IBM i.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_2.

ReceiveUserDataPtr (MQPTR):

This field specifies the address of the first *ReceiveUserData* field.

If *ReceiveExitsDefined* is greater than zero, this address is the address of the list of user data item for each channel receive exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *ReceiveExitsDefined* fields adjoining one another - one for each exit. If the number of user data items defined is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *ReceiveExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_5.

RemotePassword (MQCHAR12):

This field specifies the password from a partner.

This field contains valid information only if *ChannelType* is MQCHT_CLNTCONN or MQCHT_SVRCONN.

- For a security exit at an MQCHT_CLNTCONN channel, this password is a password which has been obtained from the environment. The exit can choose to send it to the security exit at the server.
- For a security exit at an MQCHT_SVRCONN channel, this field might contain a password which has been obtained from the environment at the client, if there is no client security exit. The exit can use this password to validate the user identifier in *RemoteUserIdentifier*.

If there is a security exit at the client, then this information can be obtained in a security flow from the client.

The length of this field is given by MQ_PASSWORD_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_3.

RemoteSecurityId (MQBYTE40):

This field specifies the security identifier for the remote user.

This field is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

The following special value indicates that there is no security identifier:

MQSID_NONE
No security identifier specified.

The value is binary zero for the length of the field.

For the C programming language, the constant MQSID_NONE_ARRAY is also defined; this constant has the same value as MQSID_NONE, but is an array of characters instead of a string.

This is an input field to the exit. The length of this field is given by MQ_SECURITY_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_6.

The following fields in this structure are not present if *Version* is less than MQCD_VERSION_7.

RemoteUserIdentifier (MQCHAR12):

This field specifies the first 12 bytes of a user identifier from a partner.

There are two fields that contain the remote user identifier:

- *RemoteUserIdentifier* contains the first 12 bytes of the remote user identifier, and is padded with blanks if the identifier is shorter than 12 bytes. *RemoteUserIdentifier* can be blank.
- *LongRemoteUserIdPtr* points to the full remote user identifier, which can be longer than 12 bytes. Its length is given by *LongRemoteUserIdLength*. The full identifier contains no trailing blanks, and is not null-terminated. If the identifier is blank, *LongRemoteUserIdLength* is zero, and the value of *LongRemoteUserIdPtr* is undefined.

  LongRemoteUserIdPtr is not present if *Version* is less than MQCD_VERSION_6.

The remote user identifier is relevant only for channels with a *ChannelType* of MQCHT_CLNTCONN or MQCHT_SVRCONN.

- For a security exit on an MQCHT_CLNTCONN channel, this value is a user identifier that has been obtained from the environment. The exit can choose to send it to the security exit at the server.
- For a security exit on an MQCHT_SVRCONN channel, this field might contain a user identifier which has been obtained from the environment at the client, if there is no client security exit. The exit might validate this user ID (possibly with the password in *RemotePassword*) and update the value in *MCAUserIdentifier*.

  If there is a security exit at the client, then this information can be obtained in a security flow from the client.

The length of this field is given by MQ_USER_ID_LENGTH. This field is not present if *Version* is less than MQCD_VERSION_2.

SecurityExit (MQCHARn):

This field specifies the channel security exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is given the opportunity to instigate security flows to validate connection authorization.
- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote machine are given to the exit.
- At initialization and termination of the channel.

See"MQCD - Channel definition" on page 2348 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

SecurityUserData (MQCHAR32):

This channel specifies the channel security exit user data.

This data is passed to the channel security exit in the *ExitData* field of the *ChannelExitParms* parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes do not effect on the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This field is not relevant in WebSphere MQ for IBM i.

SendExit (MQCHARn):

This field specifies the channel send exit name.

If this name is nonblank, the exit is called at the following times:

- Immediately before data is sent out on the network.

  The exit is given the complete transmission buffer before it is transmitted. The contents of the buffer can be modified as required.
- At initialization and termination of the channel.

See "MQCD - Channel definition" on page 2348 for a description of the content of this field in various environments.

The length of this field is given by MQ_EXIT_NAME_LENGTH.

Note: The value of this constant is environment-specific.

SendExitPtr (MQPTR):

This field specifies the address of the first *SendExit* field.

If *SendExitsDefined* is greater than zero, this address is the address of the list of names of each channel send exit in the chain.

Each name is in a field of length *ExitNameLength*, padded to the right with blanks. There are *SendExitsDefined* fields adjoining one another - one for each exit.

Any changes made to these names by an exit are preserved, although the message send exit takes no explicit action - it does not change which exits are invoked.

If *SendExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SendExitsDefined (MQLONG):

This field specifies the number of channel send exits defined in the chain.

It is greater than or equal to zero.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SendUserData (MQCHAR32):

This field specifies the channel send exit user data.

This data is passed to the channel send exit in the *ExitData* field of the *ChannelExitParms* parameter (see MQ_CHANNEL_EXIT).

This field initially contains the data that was set in the channel definition. However, during the lifetime of this MCA instance, any changes made to the contents of this field by an exit of any type are preserved by the MCA, and made visible to subsequent invocations of exits (regardless of type) for this MCA instance. This applies to exits on different conversations. Such changes do not affect the channel definition used by other MCA instances. Any characters (including binary data) can be used.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This field is not relevant in WebSphere MQ for IBM i.

SendUserDataPtr (MQPTR):

This field specifies the address of the *SendUserData* field.

If *SendExitsDefined* is greater than zero, this address is the address of the list of user data items for each channel message exit in the chain.

Each user data item is in a field of length *ExitDataLength*, padded to the right with blanks. There are *MsgExitsDefined* fields adjoining one another - one for each exit. If the number of user data items defined

is less than the number of exit names, undefined user data items are set to blanks. Conversely, if the number of user data items defined is greater than the number of exit names, the excess user data items are ignored and not presented to the exit.

Any changes made to these values by an exit are preserved. This allows one exit to pass information to another exit. No validation is carried out on any changes so, for example, binary data can be written to these fields if required.

If *SendExitsDefined* is zero, this field is the null pointer.

On platforms where the programming language does not support the pointer data type, this field is declared as a byte string of the appropriate length.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_4.

SeqNumberWrap (MQLONG):

This field specifies the highest allowable message sequence number.

When this value is reached, sequence numbers wrap to start again at 1.

This value is non-negotiable and must match in both the local and remote channel definitions.

This field is not relevant for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

SharingConversations (MQLONG):

This field specifies the maximum number of conversations that can share a channel instance associated with this channel.

This field is used on client connection and server-connection channels.

A value of 0 means that the channel operates as it did in versions earlier than WebSphere MQ Version 7.0 with respect to the following attributes:

- Conversation sharing
- Read ahead
- `STOP CHANNEL(<channelname>) MODE(QUIESCE)`
- Heartbeating
- Client asynchronous consumption

A value of 1 is the minimum value for WebSphere MQ V7.0 behavior. Although only one conversation is allowed on the channel instance, read ahead, asynchronous consumption, and the Version 7 behavior of `CLNTCONN-SVRCONN` heartbeating and quiescent channel stopping are available.

This is an input field to the exit. It is not present if *Version* is less than MQCD_VERSION_9.

The default value of this field is 10.

Note: *MaxInstances* and *MaxInstancesPerClient* limits applied to a channel restrict the number of channel instances, not the number of conversations that might be sharing those instances.

ShortConnectionName (MQCHAR20):

This field specifies the first 20 bytes of a connection name.

If the *Version* field is MQCD_VERSION_1, *ShortConnectionName* contains the full connection name.

If the *Version* field is MQCD_VERSION_2 or greater, *ShortConnectionName* contains the first 20 characters of the connection name. The full connection name is given by the *ConnectionName* field; *ShortConnectionName* and the first 20 characters of *ConnectionName* are identical.

See *ConnectionName* for details of the contents of this field.

Note: The name of this field was changed for MQCD_VERSION_2 and subsequent versions of MQCD; the field was previously called *ConnectionName*.

The length of this field is given by MQ_SHORT_CONN_NAME_LENGTH.

ShortRetryCount (MQLONG):

This field specifies the maximum number of attempts that are made to connect to a remote machine.

This field is the maximum number of attempts that are made to connect to the remote machine, at intervals specified by *ShortRetryInterval*, before the (normally longer) *LongRetryCount* and *LongRetryInterval* are used.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

ShortRetryInterval (MQLONG):

This field specifies the maximum number of seconds to wait before reattempting connection to the remote machine.

The interval between retries might be extended if the channel has to wait to become active.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

SSLCipherSpec (MQCHAR32):

This field specifies the Cipher Spec that is in use when using SSL.

If SSLCipherSpec is blank, the channel is not using SSL. If it is not blank, this field contains a string specifying the CipherSpec in use.

This parameter is valid for all channel types. It is supported on AIX, HP-UX, Linux, IBM i, Solaris, Windows, and z/OS. It is valid only for channel types of a transport type (TRPTYPE) of TCP.

This is an input field to the exit. The length of this field is given by MQ_SSL_CIPHER_SPEC_LENGTH. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLClientAuth (MQLONG):

This field specifies whether SSL client authentication is required.

This field is relevant only to SVRCONN channel definitions.

It is one of the following values:

MQSCA_REQUIRED
Client authentication required.

MQSCA_OPTIONAL
Client authentication optional.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLPeerNameLength (MQLONG):

This field specifies the length in bytes of the SSL peer name pointed to by *SSLPeerNamePtr*.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

SSLPeerNamePtr (MQPTR):

This field specifies the address of the SSL peer name.

When a certificate is received during a successful SSL handshake, the Distinguished Name of the subject of the certificate is copied into the MQCD field accessed by SSLPeerNamePtr at the end of the channel which receives the certificate. It overwrites the SSLPeerName value for the channel if this value is present in the channel definition of the local user. If a security exit is specified at this end of the channel it receives the Distinguished Name from the peer certificate in the MQCD.

This is an input field to the exit. The field is not present if *Version* is less than MQCD_VERSION_7.

Note: Security exit applications constructed prior to the release of WebSphere MQ v7.1 may require updating. For more information see Channel security exit programs.

StrucLength (MQLONG):

This field specifies the length in bytes of the MQCD structure.

The length does not include any of the strings addressed by pointer fields contained within the structure. The value is one of the following:

MQCD_LENGTH_4
Length of version-4 channel definition structure.

MQCD_LENGTH_5
Length of version-5 channel definition structure.

MQCD_LENGTH_6
Length of version-6 channel definition structure.

MQCD_LENGTH_7
Length of version-7 channel definition structure.

MQCD_LENGTH_8
Length of version-8 channel definition structure.

MQCD_LENGTH_9
Length of version-9 channel definition structure.

The following constant specifies the length of the current version:

MQCD_CURRENT_LENGTH
Length of current version of channel definition structure.

Note: These constants have values that are environment-specific.

The field is not present if *Version* is less than MQCD_VERSION_4.

TpName (MQCHAR64):

This field specifies the LU 6.2 transaction program name.

This field is relevant only if the transmission protocol (*TransportType*) is MQXPT_LU62, and the *ChannelType* is not MQCHT_SVRCONN or MQCHT_RECEIVER.

This field is always blank on platforms on which the information is contained in the communications Side Object instead.

The length of this field is given by MQ_TP_NAME_LENGTH.

TransportType (MQLONG):

This field specifies the transmission protocol to be used.

The value is not checked if the channel was initiated from the other end.

It is one of the following values:

MQXPT_LU62
LU 6.2 transport protocol.

MQXPT_TCP
TCP/IP transport protocol.

MQXPT_NETBIOS
NetBIOS transport protocol.

This value is supported in the following environments: Windows.

MQXPT_SPX
SPX transport protocol.

This value is supported in the following environments: Windows, plus WebSphere MQ clients connected to these systems.

UseDLQ (MQLONG):

This field specifies whether the dead-letter queue (or undelivered message queue) is used when messages cannot be delivered by channels.

It can contain one of the following values:

MQUSEDLQ_NO
Messages that cannot be delivered by a channel are treated as a failure. The channel either discards the message, or the channel ends, in accordance with the NPMSPEED setting.

MQUSEDLQ_YES
When the DEADQ queue manager attribute provides the name of a dead-letter queue, then it is used, else the behavior is as for NO. YES is the default value.

UserIdentifier (MQCHAR12):

This field specifies the user identifier used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This field can be nonblank only on UNIX systems and Windows, and is relevant only for channels with a *ChannelType* of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, or MQCHT_CLNTCONN. On z/OS, this field is not relevant.

The length of this field is given by MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

This field is not present when *Version* is less than MQCD_VERSION_2.

Version (MQLONG):

The Version field specifies the highest version number that you can set for the structure.

The value depends on the environment:

MQCD_VERSION_1
Version 1 channel definition structure.

MQCD_VERSION_2
Version 2 channel definition structure.

Version 2 is not used by any current IBM WebSphere MQ product.

MQCD_VERSION_3
Version 3 channel definition structure.

Version 3 is the highest that you can set the field to on MQSeries Version 2 in the following environments: HP Integrity NonStop Server, and UNIX and Linux systems not listed elsewhere.

MQCD_VERSION_4
Version 4 channel definition structure.

Version 4 is not used by any current IBM WebSphere MQ product.

MQCD_VERSION_5
Version 5 channel definition structure.

Version 5 is the highest that you can set the field to on MQSeries for OS/390 Version 5 Release 2.

MQCD_VERSION_6
Version 6 channel definition structure.

Version 6 is not the current MQCD structure version of any existing IBM WebSphere MQ product. However, a version 6 MQCD structure can be passed to MQCONNX using the ClientConnOffset or ClientConnPtr fields of the MQCNO structure.

On the distributed platforms, version 6 is the default version in the MQCD_DEFAULT and MQCD_CLIENT_CONN_DEFAULT initializers. If you want to reference the MQCD_VERSION_7, MQCD_VERSION_8, or MQCD_VERSION_9 fields of the MQCD, explicitly initialize the MQCD **Version** field to MQCD_VERSION_7, MQCD_VERSION_8, or MQCD_VERSION_9 as appropriate.

On z/OS, MQCD_VERSION_7 is the default value.

MQCD_VERSION_7
Version 7 channel definition structure.

Version 7 is the highest that you can set the field to on IBM WebSphere MQ Version 5.3 in the following environments: AIX, HP-UX, Solaris, Windows, and on IBM WebSphere MQ for z/OS Version 5.3 and Version 5.3.1. MQCD_VERSION_7 is the default value for versions of IBM WebSphere MQ for z/OS.

MQCD_VERSION_8
Version 8 channel definition structure.

Version 8 is the highest that you can set the field to on IBM WebSphere MQ Version 6.0 on all platforms.

MQCD_VERSION_9
Version 9 channel definition structure.

Version 9 is the highest that you can set the field to on IBM WebSphere MQ Version 7.0 and IBM WebSphere MQ Version 7.0.1 on all platforms.

MQCD_VERSION_10
Version 10 channel definition structure.

Version 10 is the highest that you can set the field to on IBM WebSphere MQ Version 7.1 and IBM WebSphere MQ Version 7.5 on all platforms.

Fields that exist only in the more recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCD_CURRENT_VERSION
The value set in MQCD_CURRENT_VERSION is the current version of the channel definition structure being used.

The value of MQCD_CURRENT_VERSION depends on the environment. It contains the highest value supported by the platform.

MQCD_CURRENT_VERSION is not used to initialize the default structures provided in the header, copy, and include files provided for different programming languages. The default initialization of Version depends on the platform and release.

For IBM WebSphere MQ Version 7.0 and later versions, the MQCD declarations in the header, copy, and include files are initialized to MQCD_VERSION_6. To use additional MQCD fields, applications must set the version number to MQCD_CURRENT_VERSION. If you are writing an application that is portable between several environments, you must choose a version that is supported in all the environments.

Tip: When a new version of the MQCD structure is introduced, the layout of the existing part is not changed. The exit must check the version number. It must be equal to or greater than the lowest version that contains the fields that the exit needs to use.

XmitQName (MQCHAR48):

This field specifies the name of the transmission queue from which messages are retrieved.

This field is relevant only for channels with a *ChannelType* of MQCHT_SENDER or MQCHT_SERVER.

The length of this field is given by MQ_Q_NAME_LENGTH.

C declaration:

This declaration is the C declaration for the MQCD structure.

```
typedef struct tagMQCD MQCD;
 typedef MQCD  MQPOINTER PMQCD;
 typedef PMQCD MQPOINTER PPMQCD;

 struct tagMQCD {
   MQCHAR    ChannelName[20];           /* Channel definition name */
   MQLONG    Version;                   /* Structure version number */
   MQLONG    ChannelType;               /* Channel type */
   MQLONG    TransportType;             /* Transport type */
   MQCHAR    Desc[64];                  /* Channel description */
   MQCHAR    QMgrName[48];              /* Queue-manager name */
   MQCHAR    XmitQName[48];             /* Transmission queue name */
   MQCHAR    ShortConnectionName[20];   /* First 20 bytes of */
                                        /* connection name */
   MQCHAR    MCAName[20];               /* Reserved */
   MQCHAR    ModeName[8];               /* LU 6.2 Mode name */
   MQCHAR    TpName[64];                /* LU 6.2 transaction program */
                                        /* name */
   MQLONG    BatchSize;                 /* Batch size */
   MQLONG    DiscInterval;              /* Disconnect interval */
   MQLONG    ShortRetryCount;           /* Short retry count */
   MQLONG    ShortRetryInterval;        /* Short retry wait interval */
   MQLONG    LongRetryCount;            /* Long retry count */
   MQLONG    LongRetryInterval;         /* Long retry wait interval */
   MQCHAR    SecurityExit[128];         /* Channel security exit name */
   MQCHAR    MsgExit[128];              /* Channel message exit name */
   MQCHAR    SendExit[128];             /* Channel send exit name */
   MQCHAR    ReceiveExit[128];          /* Channel receive exit name */
   MQLONG    SeqNumberWrap;             /* Highest allowable message */
                                        /* sequence number */
   MQLONG    MaxMsgLength;              /* Maximum message length */
   MQLONG    PutAuthority;              /* Put authority */
   MQLONG    DataConversion;            /* Data conversion */
   MQCHAR    SecurityUserData[32];      /* Channel security exit user */
                                        /* data */
   MQCHAR    MsgUserData[32];           /* Channel message exit user */
                                        /* data */
   MQCHAR    SendUserData[32];          /* Channel send exit user */
                                        /* data */
   MQCHAR    ReceiveUserData[32];       /* Channel receive exit user */
                                        /* data */
   /* Ver:1 */
   MQCHAR    UserIdentifier[12];        /* User identifier */
   MQCHAR    Password[12];              /* Password */
   MQCHAR    MCAUserIdentifier[12];     /* First 12 bytes of MCA user */
                                        /* identifier */
   MQLONG    MCAType;                   /* Message channel agent type */
   MQCHAR    ConnectionName[264];       /* Connection name */
   MQCHAR    RemoteUserIdentifier[12];  /* First 12 bytes of user */
                                        /* identifier from partner */
   MQCHAR    RemotePassword[12];        /* Password from partner */
   /* Ver:2 */
   MQCHAR    MsgRetryExit[128];         /* Channel message retry exit */
                                        /* name */
```

```
MQCHAR    MsgRetryUserData[32];      /* Channel message retry exit */
                                     /* user data */
MQLONG    MsgRetryCount;             /* Number of times MCA will */
                                     /* try to put the message, */
                                     /* after first attempt has */
                                     /* failed */
MQLONG    MsgRetryInterval;          /* Minimum interval in */
                                     /* milliseconds after which */
                                     /* the open or put operation */
                                     /* will be retried */
/* Ver:3 */
MQLONG    HeartbeatInterval;         /* Time in seconds between */
                                     /* heartbeat flows */
MQLONG    BatchInterval;             /* Batch duration */
MQLONG    NonPersistentMsgSpeed;     /* Speed at which */
                                     /* nonpersistent messages are */
                                     /* sent */
MQLONG    StrucLength;               /* Length of MQCD structure */
MQLONG    ExitNameLength;            /* Length of exit name */
MQLONG    ExitDataLength;            /* Length of exit user data */
MQLONG    MsgExitsDefined;           /* Number of message exits */
                                     /* defined */
MQLONG    SendExitsDefined;          /* Number of send exits */
                                     /* defined */
MQLONG    ReceiveExitsDefined;       /* Number of receive exits */
                                     /* defined */
MQPTR     MsgExitPtr;                /* Address of first MsgExit */
                                     /* field */
MQPTR     MsgUserDataPtr;            /* Address of first */
                                     /* MsgUserData field */
MQPTR     SendExitPtr;               /* Address of first SendExit */
                                     /* field */
MQPTR     SendUserDataPtr;           /* Address of first */
                                     /* SendUserData field */
MQPTR     ReceiveExitPtr;            /* Address of first */
                                     /* ReceiveExit field */
MQPTR     ReceiveUserDataPtr;        /* Address of first */
                                     /* ReceiveUserData field */
/* Ver:4 */
MQPTR     ClusterPtr;                /* Address of a list of */
                                     /* cluster names */
MQLONG    ClustersDefined;           /* Number of clusters to */
                                     /* which the channel belongs */
MQLONG    NetworkPriority;           /* Network priority */
/* Ver:5 */
MQLONG    LongMCAUserIdLength;       /* Length of long MCA user */
                                     /* identifier */
MQLONG    LongRemoteUserIdLength;    /* Length of long remote user */
                                     /* identifier */
MQPTR     LongMCAUserIdPtr;          /* Address of long MCA user */
                                     /* identifier */
MQPTR     LongRemoteUserIdPtr;       /* Address of long remote */
                                     /* user identifier */
MQBYTE40  MCASecurityId;             /* MCA security identifier */
MQBYTE40  RemoteSecurityId;          /* Remote security identifier */
/* Ver:6 */
MQCHAR    SSLCipherSpec[32];         /* SSL CipherSpec */
MQPTR     SSLPeerNamePtr;            /* Address of SSL peer name */
MQLONG    SSLPeerNameLength;         /* Length of SSL peer name */
MQLONG    SSLClientAuth;             /* Whether SSL client */
                                     /* authentication is required */
MQLONG    KeepAliveInterval;         /* Keepalive interval */
MQCHAR    LocalAddress[48];          /* Local communications */
                                     /* address */
MQLONG    BatchHeartbeat;            /* Batch heartbeat interval */
/* Ver:7 */
MQLONG    HdrCompList[2];            /* Header data compression */
```

```
                                    /* list */
  MQLONG    MsgCompList[16];        /* Message data compression */
                                    /* list */
  MQLONG    CLWLChannelRank;        /* Channel rank */
  MQLONG    CLWLChannelPriority;    /* Channel priority */
  MQLONG    CLWLChannelWeight;      /* Channel weight */
  MQLONG    ChannelMonitoring;      /* Channel monitoring */
  MQLONG    ChannelStatistics;      /* Channel statistics */
  /* Ver:8 */
  MQLONG    SharingConversations;   /* Limit on sharing */
                                    /* conversations */
  MQLONG    PropertyControl;        /* Message property control */
  MQLONG    MaxInstances;           /* Limit on SVRCONN channel */
                                    /* instances */
  MQLONG    MaxInstancesPerClient;  /* Limit on SVRCONN channel */
                                    /* instances per client */
  MQLONG    ClientChannelWeight;    /* Client channel weight */
  MQLONG    ConnectionAffinity;     /* Connection affinity */
  /* Ver:9 */
  MQLONG    BatchDataLimit;         /* Batch data limit */
  MQLONG    UseDLQ;                 /* Use Dead Letter Queue */
  MQLONG    DefReconnect;           /* Default client reconnect */
                                    /* option */
  /* Ver:10 */
};
```

COBOL declaration:

This declaration is the COBOL declaration for the MQCD structure.

```
** MQCD structure
       10  MQCD.
      ** Channel definition name
       15  MQCD-CHANNELNAME PIC X(20).
      ** Structure version number
       15  MQCD-VERSION PIC S9(9) BINARY.
      ** Channel type
       15  MQCD-CHANNELTYPE PIC S9(9) BINARY.
      ** Transport type
       15  MQCD-TRANSPORTTYPE PIC S9(9) BINARY.
      ** Channel description
       15  MQCD-DESC PIC X(64).
      ** Queue-manager name
       15  MQCD-QMGRNAME PIC X(48).
      ** Transmission queue name
       15  MQCD-XMITQNAME PIC X(48).
      ** First 20 bytes of connection name
       15  MQCD-SHORTCONNECTIONNAME PIC X(20).
      ** Reserved
       15  MQCD-MCANAME PIC X(20).
      ** LU 6.2 Mode name
       15  MQCD-MODENAME PIC X(8).
      ** LU 6.2 transaction program name
       15  MQCD-TPNAME PIC X(64).
      ** Batch size
       15  MQCD-BATCHSIZE PIC S9(9) BINARY.
      ** Disconnect interval
       15  MQCD-DISCINTERVAL PIC S9(9) BINARY.
      ** Short retry count
       15  MQCD-SHORTRETRYCOUNT PIC S9(9) BINARY.
      ** Short retry wait interval
       15  MQCD-SHORTRETRYINTERVAL PIC S9(9) BINARY.
      ** Long retry count
       15  MQCD-LONGRETRYCOUNT PIC S9(9) BINARY.
      ** Long retry wait interval
       15  MQCD-LONGRETRYINTERVAL PIC S9(9) BINARY.
      ** Channel security exit name
```

```
 15  MQCD-SECURITYEXIT PIC X(20).
** Channel message exit name
 15  MQCD-MSGEXIT PIC X(20).
** Channel send exit name
 15  MQCD-SENDEXIT PIC X(20).
** Channel receive exit name
 15  MQCD-RECEIVEEXIT PIC X(20).
** Highest allowable message sequence number
 15  MQCD-SEQNUMBERWRAP PIC S9(9) BINARY.
** Maximum message length
 15  MQCD-MAXMSGLENGTH PIC S9(9) BINARY.
** Put authority
 15  MQCD-PUTAUTHORITY PIC S9(9) BINARY.
** Data conversion
 15  MQCD-DATACONVERSION PIC S9(9) BINARY.
** Channel security exit user data
 15  MQCD-SECURITYUSERDATA PIC X(32).
** Channel message exit user data
 15  MQCD-MSGUSERDATA PIC X(32).
** Channel send exit user data
 15  MQCD-SENDUSERDATA PIC X(32).
** Channel receive exit user data
 15  MQCD-RECEIVEUSERDATA PIC X(32).
** Ver:1 **
** User identifier
 15  MQCD-USERIDENTIFIER PIC X(12).
** Password
 15  MQCD-PASSWORD PIC X(12).
** First 12 bytes of MCA user identifier
 15  MQCD-MCAUSERIDENTIFIER PIC X(12).
** Message channel agent type
 15  MQCD-MCATYPE PIC S9(9) BINARY.
** Connection name
 15  MQCD-CONNECTIONNAME PIC X(264).
** First 12 bytes of user identifier from partner
 15  MQCD-REMOTEUSERIDENTIFIER PIC X(12).
** Password from partner
 15  MQCD-REMOTEPASSWORD PIC X(12).
** Ver:2 **
** Channel message retry exit name
 15  MQCD-MSGRETRYEXIT PIC X(20).
** Channel message retry exit user data
 15  MQCD-MSGRETRYUSERDATA PIC X(32).
** Number of times MCA will try to put the message, after first
** attempt has failed
 15  MQCD-MSGRETRYCOUNT PIC S9(9) BINARY.
** Minimum interval in milliseconds after which the open or put
** operation will be retried
 15  MQCD-MSGRETRYINTERVAL PIC S9(9) BINARY.
** Ver:3 **
** Time in seconds between heartbeat flows
 15  MQCD-HEARTBEATINTERVAL PIC S9(9) BINARY.
** Batch duration
 15  MQCD-BATCHINTERVAL PIC S9(9) BINARY.
** Speed at which nonpersistent messages are sent
 15  MQCD-NONPERSISTENTMSGSPEED PIC S9(9) BINARY.
** Length of MQCD structure
 15  MQCD-STRUCLENGTH PIC S9(9) BINARY.
** Length of exit name
 15  MQCD-EXITNAMELENGTH PIC S9(9) BINARY.
** Length of exit user data
 15  MQCD-EXITDATALENGTH PIC S9(9) BINARY.
** Number of message exits defined
 15  MQCD-MSGEXITSDEFINED PIC S9(9) BINARY.
** Number of send exits defined
 15  MQCD-SENDEXITSDEFINED PIC S9(9) BINARY.
** Number of receive exits defined
```

```
 15  MQCD-RECEIVEEXITSDEFINED PIC S9(9) BINARY.
** Address of first MsgExit field
 15  MQCD-MSGEXITPTR POINTER.
** Address of first MsgUserData field
 15  MQCD-MSGUSERDATAPTR POINTER.
** Address of first SendExit field
 15  MQCD-SENDEXITPTR POINTER.
** Address of first SendUserData field
 15  MQCD-SENDUSERDATAPTR POINTER.
** Address of first ReceiveExit field
 15  MQCD-RECEIVEEXITPTR POINTER.
** Address of first ReceiveUserData field
 15  MQCD-RECEIVEUSERDATAPTR POINTER.
** Ver:4 **
** Address of a list of cluster names
 15  MQCD-CLUSTERPTR POINTER.
** Number of clusters to which the channel belongs
 15  MQCD-CLUSTERSDEFINED PIC S9(9) BINARY.
** Network priority
 15  MQCD-NETWORKPRIORITY PIC S9(9) BINARY.
** Ver:5 **
** Length of long MCA user identifier
 15  MQCD-LONGMCAUSERIDLENGTH PIC S9(9) BINARY.
** Length of long remote user identifier
 15  MQCD-LONGREMOTEUSERIDLENGTH PIC S9(9) BINARY.
** Address of long MCA user identifier
 15  MQCD-LONGMCAUSERIDPTR POINTER.
** Address of long remote user identifier
 15  MQCD-LONGREMOTEUSERIDPTR POINTER.
** MCA security identifier
 15  MQCD-MCASECURITYID PIC X(40).
** Remote security identifier
 15  MQCD-REMOTESECURITYID PIC X(40).
** Ver:6 **
** SSL CipherSpec
 15  MQCD-SSLCIPHERSPEC PIC X(32).
** Address of SSL peer name
 15  MQCD-SSLPEERNAMEPTR POINTER.
** Length of SSL peer name
 15  MQCD-SSLPEERNAMELENGTH PIC S9(9) BINARY.
** Whether SSL client authentication is required
 15  MQCD-SSLCLIENTAUTH PIC S9(9) BINARY.
** Keepalive interval
 15  MQCD-KEEPALIVEINTERVAL PIC S9(9) BINARY.
** Local communications address
 15  MQCD-LOCALADDRESS PIC X(48).
** Batch heartbeat interval
 15  MQCD-BATCHHEARTBEAT PIC S9(9) BINARY.
** Ver:7 **
** Header data compression list
 15  MQCD-HDRCOMPLIST PIC S9(9) BINARY.
** Message data compression list
 15  MQCD-MSGCOMPLIST PIC S9(9) BINARY.
** Channel rank
 15  MQCD-CLWLCHANNELRANK PIC S9(9) BINARY.
** Channel priority
 15  MQCD-CLWLCHANNELPRIORITY PIC S9(9) BINARY.
** Channel weight
 15  MQCD-CLWLCHANNELWEIGHT PIC S9(9) BINARY.
** Channel monitoring
 15  MQCD-CHANNELMONITORING PIC S9(9) BINARY.
** Channel statistics
 15  MQCD-CHANNELSTATISTICS PIC S9(9) BINARY.
** Ver:8 **
** Limit on sharing conversations
 15  MQCD-SHARINGCONVERSATIONS PIC S9(9) BINARY.
** Message property control
```

```
        15  MQCD-PROPERTYCONTROL PIC S9(9) BINARY.
       ** Limit on SVRCONN channel instances
        15  MQCD-MAXINSTANCES PIC S9(9) BINARY.
       ** Limit on SVRCONN channel instances per client
        15  MQCD-MAXINSTANCESPERCLIENT PIC S9(9) BINARY.
       ** Client channel weight
        15  MQCD-CLIENTCHANNELWEIGHT PIC S9(9) BINARY.
       ** Connection affinity
        15  MQCD-CONNECTIONAFFINITY PIC S9(9) BINARY.
       ** Ver:9 **
       ** Batch data limit
        15  MQCD-BATCHDATALIMIT PIC S9(9) BINARY.
       ** Use Dead Letter Queue
        15  MQCD-USEDLQ PIC S9(9) BINARY.
       ** Default client reconnect option
        15  MQCD-DEFRECONNECT PIC S9(9) BINARY.
       ** Ver:10 **
```

RPG declaration (ILE):

This declaration is the RPG declaration for the MQCD structure.

```
D* MQCD Structure
     D*
     D* Channel definition name
     D  CDCHN                  1     20
     D* Structure version number
     D  CDVER                 21     24I 0
     D* Channel type
     D  CDCHT                 25     28I 0
     D* Transport type
     D  CDTRT                 29     32I 0
     D* Channel description
     D  CDDES                 33     96
     D* Queue-manager name
     D  CDQM                  97    144
     D* Transmission queue name
     D  CDXQ                 145    192
     D* First 20 bytes of connection name
     D  CDSCN                193    212
     D* Reserved
     D  CDMCA                213    232
     D* LU 6.2 Mode name
     D  CDMOD                233    240
     D* LU 6.2 transaction program name
     D  CDTP                 241    304
     D* Batch size
     D  CDBS                 305    308I 0
     D* Disconnect interval
     D  CDDI                 309    312I 0
     D* Short retry count
     D  CDSRC                313    316I 0
     D* Short retry wait interval
     D  CDSRI                317    320I 0
     D* Long retry count
     D  CDLRC                321    324I 0
     D* Long retry wait interval
     D  CDLRI                325    328I 0
     D* Channel security exit name
     D  CDSCX                329    348
     D* Channel message exit name
     D  CDMSX                349    368
     D* Channel send exit name
     D  CDSNX                369    388
     D* Channel receive exit name
     D  CDRCX                389    408
     D* Highest allowable message sequence number
```

```
D  CDSNW                409    412I 0
D* Maximum message length
D  CDMML                413    416I 0
D* Put authority
D  CDPA                 417    420I 0
D* Data conversion
D  CDDC                 421    424I 0
D* Channel security exit user data
D  CDSCD                425    456
D* Channel message exit user data
D  CDMSD                457    488
D* Channel send exit user data
D  CDSND                489    520
D* Channel receive exit user data
D  CDRCD                521    552
D* Ver:1 **
D* User identifier
D  CDUID                553    564
D* Password
D  CDPW                 565    576
D* First 12 bytes of MCA user identifier
D  CDAUI                577    588
D* Message channel agent type
D  CDCAT                589    592I 0
D* Connection name
D  CDCON                593    848
D  CDCN2                849    856
D* First 12 bytes of user identifier from partner
D  CDRUI                857    868
D* Password from partner
D  CDRPW                869    880
D* Ver:2 **
D* Channel message retry exit name
D  CDMRX                881    900
D* Channel message retry exit user data
D  CDMRD                901    932
D* Number of times MCA will try to put the message, after first
D* attempt has failed
D  CDMRC                933    936I 0
D* Minimum interval in milliseconds after which the open or put
D* operation will be retried
D  CDMRI                937    940I 0
D* Ver:3 **
D* Time in seconds between heartbeat flows
D  CDHBI                941    944I 0
D* Batch duration
D  CDBI                 945    948I 0
D* Speed at which nonpersistent messages are sent
D  CDNPM                949    952I 0
D* Length of MQCD structure
D  CDLEN                953    956I 0
D* Length of exit name
D  CDXNL                957    960I 0
D* Length of exit user data
D  CDXDL                961    964I 0
D* Number of message exits defined
D  CDMXD                965    968I 0
D* Number of send exits defined
D  CDSXD                969    972I 0
D* Number of receive exits defined
D  CDRXD                973    976I 0
D* Address of first MsgExit field
D  CDMXP                977    992*
D* Address of first MsgUserData field
D  CDMUP                993   1008*
D* Address of first SendExit field
D  CDSXP               1009   1024*
```

```
D* Address of first SendUserData field
D  CDSUP               1025   1040*
D* Address of first ReceiveExit field
D  CDRXP               1041   1056*
D* Address of first ReceiveUserData field
D  CDRUP               1057   1072*
D* Ver:4 **
D* Address of a list of cluster names
D  CDCLP               1073   1088*
D* Number of clusters to which the channel belongs
D  CDCLD               1089   1092I 0
D* Network priority
D  CDNP                1093   1096I 0
D* Ver:5 **
D* Length of long MCA user identifier
D  CDLML               1097   1100I 0
D* Length of long remote user identifier
D  CDLRL               1101   1104I 0
D* Address of long MCA user identifier
D  CDLMP               1105   1120*
D* Address of long remote user identifier
D  CDLRP               1121   1136*
D* MCA security identifier
D  CDMSI               1137   1176
D* Remote security identifier
D  CDRSI               1177   1216
D* Ver:6 **
D* SSL CipherSpec
D  CDSCS               1217   1248
D* Address of SSL peer name
D  CDSPN               1249   1264*
D* Length of SSL peer name
D  CDSPL               1265   1268I 0
D* Whether SSL client authentication is required
D  CDSCA               1269   1272I 0
D* Keepalive interval
D  CDKAI               1273   1276I 0
D* Local communications address
D  CDLOA               1277   1324
D* Batch heartbeat interval
D  CDBHB               1325   1328I 0
D* Ver:7 **
D* Header data compression list
D  CDHCL0
D  CDHCL1              1329   1332I 0
D  CDHCL2              1333   1336I 0
D  CDHCL                        10I 0 DIM(2) OVERLAY(CDHCL0)
D* Message data compression list
D  CDMCL0
D  CDMCL1              1337   1340I 0
D  CDMCL2              1341   1344I 0
D  CDMCL3              1345   1348I 0
D  CDMCL4              1349   1352I 0
D  CDMCL5              1353   1356I 0
D  CDMCL6              1357   1360I 0
D  CDMCL7              1361   1364I 0
D  CDMCL8              1365   1368I 0
D  CDMCL9              1369   1372I 0
D  CDMCL10             1373   1376I 0
D  CDMCL11             1377   1380I 0
D  CDMCL12             1381   1384I 0
D  CDMCL13             1385   1388I 0
D  CDMCL14             1389   1392I 0
D  CDMCL15             1393   1396I 0
D  CDMCL16             1397   1400I 0
D  CDMCL                        10I 0 DIM(16) OVERLAY(CDMCL0)
D* Channel rank
```

```
     D  CDCWCR              1401   1404I 0
     D* Channel priority
     D  CDCWCP              1405   1408I 0
     D* Channel weight
     D  CDCWCW              1409   1412I 0
     D* Channel monitoring
     D  CDCHLMON            1413   1416I 0
     D* Channel statistics
     D  CDCHLST             1417   1420I 0
     D* Ver:8 **
     D* Limit on sharing conversations
     D  CDSHC               1421   1424I 0
     D* Message property control
     D  CDPRC               1425   1428I 0
     D* Limit on SVRCONN channel instances
     D  CDMXIN              1429   1432I 0
     D* Limit on SVRCONN channel instances per client
     D  CDMXIC              1433   1436I 0
     D* Client channel weight
     D  CDCLNCHLW           1437   1440I 0
     D* Connection affinity
     D  CDCONNAFF           1441   1444I 0
     D* Ver:9 **
     D* Batch data limit
     D  CDBDL               1445   1448I 0
     D* Use Dead Letter Queue
     D  CDUDLQ              1449   1452I 0
     D* Default client reconnect option
     D  CDDRCN              1453   1456I 0
     D* Ver:10 **
```

System/390 assembler declaration:

This declaration is the System/390 assembler declaration for the MQCD structure.

```
MQCD                        DSECT
MQCD_CHANNELNAME            DS   CL20   Channel definition name
MQCD_VERSION                DS   F      Structure version number
MQCD_CHANNELTYPE            DS   F      Channel type
MQCD_TRANSPORTTYPE          DS   F      Transport type
MQCD_DESC                   DS   CL64   Channel description
MQCD_QMGRNAME               DS   CL48   Queue-manager name
MQCD_XMITQNAME              DS   CL48   Transmission queue name
MQCD_SHORTCONNECTIONNAME    DS   CL20   First 20 bytes of connection
*                                       name
MQCD_MCANAME                DS   CL20   Reserved
MQCD_MODENAME               DS   CL8    LU 6.2 Mode name
MQCD_TPNAME                 DS   CL64   LU 6.2 transaction program name
MQCD_BATCHSIZE              DS   F      Batch size
MQCD_DISCINTERVAL           DS   F      Disconnect interval
MQCD_SHORTRETRYCOUNT        DS   F      Short retry count
MQCD_SHORTRETRYINTERVAL     DS   F      Short retry wait interval
MQCD_LONGRETRYCOUNT         DS   F      Long retry count
MQCD_LONGRETRYINTERVAL      DS   F      Long retry wait interval
MQCD_SECURITYEXIT           DS   CLn    Channel security exit name
MQCD_MSGEXIT                DS   CLn    Channel message exit name
MQCD_SENDEXIT               DS   CLn    Channel send exit name
MQCD_RECEIVEEXIT            DS   CLn    Channel receive exit name
MQCD_SEQNUMBERWRAP          DS   F      Highest allowable message
*                                       sequence number
MQCD_MAXMSGLENGTH           DS   F      Maximum message length
MQCD_PUTAUTHORITY           DS   F      Put authority
MQCD_DATACONVERSION         DS   F      Data conversion
MQCD_SECURITYUSERDATA       DS   CL32   Channel security exit user data
MQCD_MSGUSERDATA            DS   CL32   Channel message exit user data
MQCD_SENDUSERDATA           DS   CL32   Channel send exit user data
MQCD_RECEIVEUSERDATA        DS   CL32   Channel receive exit user data
```

```
MQCD_USERIDENTIFIER          DS   CL12   User identifier
MQCD_PASSWORD                DS   CL12   Password
MQCD_MCAUSERIDENTIFIER       DS   CL12   First 12 bytes of MCA user
*                                        identifier
MQCD_MCATYPE                 DS   F      Message channel agent type
MQCD_CONNECTIONNAME          DS   CL264  Connection name
MQCD_REMOTEUSERIDENTIFIER    DS   CL12   First 12 bytes of user
*                                        identifier from partner
MQCD_REMOTEPASSWORD          DS   CL12   Password from partner
MQCD_MSGRETRYEXIT            DS   CLn    Channel message retry exit name
MQCD_MSGRETRYUSERDATA        DS   CL32   Channel message retry exit user
*                                        data
MQCD_MSGRETRYCOUNT           DS   F      Number of times MCA will try to
*                                        put the message, after the
*                                        first attempt has failed
MQCD_MSGRETRYINTERVAL        DS   F      Minimum interval in
*                                        milliseconds after which the
*                                        open or put operation will be
*                                        retried
MQCD_HEARTBEATINTERVAL       DS   F      Time in seconds between
*                                        heartbeat flows
MQCD_BATCHINTERVAL           DS   F      Batch duration
MQCD_NONPERSISTENTMSGSPEED   DS   F      Speed at which nonpersistent
*                                        messages are sent
MQCD_STRUCLENGTH             DS   F      Length of MQCD structure
MQCD_EXITNAMELENGTH          DS   F      Length of exit name
MQCD_EXITDATALENGTH          DS   F      Length of exit user data
MQCD_MSGEXITSDEFINED         DS   F      Number of message exits defined
MQCD_SENDEXITSDEFINED        DS   F      Number of send exits defined
MQCD_RECEIVEEXITSDEFINED     DS   F      Number of receive exits defined
MQCD_MSGEXITPTR              DS   F      Address of first MSGEXIT field
MQCD_MSGUSERDATAPTR          DS   F      Address of first MSGUSERDATA
*                                        field
MQCD_SENDEXITPTR             DS   F      Address of first SENDEXIT field
MQCD_SENDUSERDATAPTR         DS   F      Address of first SENDUSERDATA
*                                        field
MQCD_RECEIVEEXITPTR          DS   F      Address of first RECEIVEEXIT
*                                        field
MQCD_RECEIVEUSERDATAPTR      DS   F      Address of first
*                                        RECEIVEUSERDATA field
MQCD_CLUSTERPTR              DS   F      Address of a list of cluster
*                                        names
MQCD_CLUSTERSDEFINED         DS   F      Number of clusters to which the
*                                        channel belongs
MQCD_NETWORKPRIORITY         DS   F      Network priority
MQCD_LONGMCAUSERIDLENGTH     DS   F      Length of long MCA user
*                                        identifier
MQCD_LONGREMOTEUSERIDLENGTH  DS   F      Length of long remote user
*                                        identifier
MQCD_LONGMCAUSERIDPTR        DS   F      Address of long MCA user
*                                        identifier
MQCD_LONGREMOTEUSERIDPTR     DS   F      Address of long remote user
*                                        identifier
MQCD_MCASECURITYID           DS   XL40   MCA security identifier
MQCD_REMOTESECURITYID        DS   XL40   Remote security identifier
MQCD_SSLCIPHERSPEC           DS   CL32   SSL CipherSpec
MQCD_SSLPEERNAMEPTR          DS   F      Address of SSL peer name
MQCD_SSLPEERNAMELENGTH       DS   F      Length of SSL peer name
MQCD_SSLCLIENTAUTH           DS   F      Whether SSL client
*                                        authentication is required
MQCD_KEEPALIVEINTERVAL       DS   F      Keepalive interval
MQCD_LOCALADDRESS            DS   CL48   Local communications address
MQCD_BATCHHEARTBEAT          DS   F      Batch heartbeat interval
MQCD_HDRCOMPLIST             DS   CL2    Header data compression list
MQCD_MSGCOMPLIST             DS   CL16   Message data compression list
MQCD_CLWLCHANNELRANK         DS   F      Channel rank
MQCD_CLWLCHANNELPRIORITY     DS   F      Channel priority
```

```
MQCD_CLWLCHANNELWEIGHT       DS   F      Channel weight
MQCD_CHANNELMONITORING       DS   F      Channel monitoring
MQCD_CHANNELSTATISTICS       DS   F      Channel statistics
MQCD_SHARINGCONVERSATIONS    DS   F      Limit on sharing
*                                        conversations
MQCD_PROPERTYCONTROL         DS   F      Message property
*                                        control
MQCD_SHARINGCONVERSATIONS    DS   F      Limit on sharing conversations
MQCD_PROPERTYCONTROL         DS   F      Message property control
MQCD_MAXINSTANCES            DS   F      Limit on SVRCONN chl instances
MQCD_MAXINSTANCESPERCLIENT   DS   F      Limit on SVRCONN chl instances
                                         per client
MQCD_CLIENTCHANNELWEIGHT     DS   F      Channel weight
MQCD_CONNECTIONAFFINITY      DS   F      Connection Affinty
MQCD_BATCHDATALIMIT          DS   F      Batch data limit
MQCD_USEDLQ                  DS   F      Use dead-letter queue
MQCD_DEFRECONNECT            DS   F      Default client reconnect option
MQCD_LENGTH                  EQU  *-MQCD
                             ORG  MQCD
MQCD_AREA                    DS   CL(MQCD_LENGTH)
```

Visual Basic declaration:

This declaration is the Visual Basic declaration of the MQCD structure.

In Visual Basic, the MQCD structure can be used with the MQCNO structure on the MQCONNX call.

```
Type MQCD
  ChannelName            As String*20  'Channel definition name'
  Version                As Long       'Structure version number'
  ChannelType            As Long       'Channel type'
  TransportType          As Long       'Transport type'
  Desc                   As String*64  'Channel description'
  QMgrName               As String*48  'Queue-manager name'
  XmitQName              As String*48  'Transmission queue name'
  ShortConnectionName    As String*20  'First 20 bytes of connection'
                                       'name'
  MCAName                As String*20  'Reserved'
  ModeName               As String*8   'LU 6.2 Mode name'
  TpName                 As String*64  'LU 6.2 transaction program name'
  BatchSize              As Long       'Batch size'
  DiscInterval           As Long       'Disconnect interval'
  ShortRetryCount        As Long       'Short retry count'
  ShortRetryInterval     As Long       'Short retry wait interval'
  LongRetryCount         As Long       'Long retry count'
  LongRetryInterval      As Long       'Long retry wait interval'
  SecurityExit           As String*128 'Channel security exit name'
  MsgExit                As String*128 'Channel message exit name'
  SendExit               As String*128 'Channel send exit name'
  ReceiveExit            As String*128 'Channel receive exit name'
  SeqNumberWrap          As Long       'Highest allowable message'
                                       'sequence number'
  MaxMsgLength           As Long       'Maximum message length'
  PutAuthority           As Long       'Put authority'
  DataConversion         As Long       'Data conversion'
  SecurityUserData       As String*32  'Channel security exit user data'
  MsgUserData            As String*32  'Channel message exit user data'
  SendUserData           As String*32  'Channel send exit user data'
  ReceiveUserData        As String*32  'Channel receive exit user data'
  UserIdentifier         As String*12  'User identifier'
  Password               As String*12  'Password'
  MCAUserIdentifier      As String*12  'First 12 bytes of MCA user'
                                       'identifier'
  MCAType                As Long       'Message channel agent type'
  ConnectionName         As String*264 'Connection name'
  RemoteUserIdentifier   As String*12  'First 12 bytes of user'
                                       'identifier from partner'
```

```
  RemotePassword         As String*12  'Password from partner'
  MsgRetryExit           As String*128 'Channel message retry exit name'
  MsgRetryUserData       As String*32  'Channel message retry exit user'
                                       'data'
  MsgRetryCount          As Long       'Number of times MCA will try to'
                                       'put the message, after the'
                                       'first attempt has failed'
  MsgRetryInterval       As Long       'Minimum interval in'
                                       'milliseconds after which the'
                                       'open or put operation will be'
                                       'retried'
  HeartbeatInterval      As Long       'Time in seconds between'
                                       'heartbeat flows'
  BatchInterval          As Long       'Batch duration'
  NonPersistentMsgSpeed  As Long       'Speed at which nonpersistent'
                                       'messages are sent'
  StrucLength            As Long       'Length of MQCD structure'
  ExitNameLength         As Long       'Length of exit name'
  ExitDataLength         As Long       'Length of exit user data'
  MsgExitsDefined        As Long       'Number of message exits defined'
  SendExitsDefined       As Long       'Number of send exits defined'
  ReceiveExitsDefined    As Long       'Number of receive exits defined'
  MsgExitPtr             As MQPTR      'Address of first MsgExit field'
  MsgUserDataPtr         As MQPTR      'Address of first MsgUserData'
                                       'field'
  SendExitPtr            As MQPTR      'Address of first SendExit field'
  SendUserDataPtr        As MQPTR      'Address of first SendUserData'
                                       'field'
  ReceiveExitPtr         As MQPTR      'Address of first ReceiveExit'
                                       'field'
  ReceiveUserDataPtr     As MQPTR      'Address of first'
                                       'ReceiveUserData field'
  ClusterPtr             As MQPTR      'Address of a list of cluster'
                                       'names'
  ClustersDefined        As Long       'Number of clusters to which the'
                                       'channel belongs'
  NetworkPriority        As Long       'Network priority'
  LongMCAUserIdLength    As Long       'Length of long MCA user'
                                       'identifier'
  LongRemoteUserIdLength As Long       'Length of long remote user'
                                       'identifier'
  LongMCAUserIdPtr       As MQPTR      'Address of long MCA user'
                                       'identifier'
  LongRemoteUserIdPtr    As MQPTR      'Address of long remote user'
                                       'identifier'
  MCASecurityId          As MQBYTE40   'MCA security identifier'
  RemoteSecurityId       As MQBYTE40   'Remote security identifier'
  SSLCipherSpec          As String*32  'SSL CipherSpec'
  SSLPeerNamePtr         As MQPTR      'Address of SSL peer name'
  SSLPeerNameLength      As Long       'Length of SSL peer name'
  SSLClientAuth          As Long       'Whether SSL client'
                                       'authentication is required'
  KeepAliveInterval      As Long       'Keepalive interval'
  LocalAddress           As String*48  'Local communications address'
  BatchHeartbeat         As Long       'Batch heartbeat interval'
  HdrCompList(0 to 1)    As Long2      'Header data compression list'
  MsgCompList(0 To 15)   As Long16     'Message data compression list'
  CLWLChannelRank        As Long       'Channel Rank'
  CLWLChannelPriority    As Long       'Channel priority'
  CLWLChannelWeight      As Long       'Channel Weight'
  ChannelMonitoring      As Long       'Channel Monitoring control'
  ChannelStatistics      As Long       'Channel Statistics'
End Type
```

Changing MQCD fields in a channel exit:

A channel exit can change fields in the MQCD. However, these changes are not typically acted on, except in the circumstances listed.

If a channel exit program changes a field in the MQCD data structure, the new value is typically ignored by the WebSphere MQ channel process. However, the new value remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance.

If SharingConversations is set to FALSE in the MQCXP structure, changes to certain fields can be acted on, depending on the type of exit program, the type of channel, and the exit reason code. The following table shows the fields that can be changed and affect the behavior of the channel, and in what circumstances. If an exit program changes one of these fields in any other circumstances, or any field not listed, the new value is ignored by the channel process. The new value remains in the MQCD and is passed to any remaining exits in an exit chain and to any conversation sharing the channel instance.

Any type of exit program when called for initialization (MQXR_INIT) can change the ChannelName field of any type of channel, as long as MQCXP SharingConverstions is set to FALSE. Only a security exit can change the MCAUserIdentifier field, regardless of the value of MQCXP SharingConverstions.

Field	**Exit Reason Code**	**Exit Type**	**Channel Type**
ChannelName	MQXR_INIT	All	All
TransportType	MQXR_INIT	All	All
XmitQName	MQXR_INIT	All	SDR, RCVR
ModeName	MQXR_INIT	All	All
TpName	MQXR_INIT	All	All
BatchSize	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
DiscInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
ShortRetryCount	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
ShortRetryInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
LongRetryCount	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR

Field	Exit Reason Code	Exit Type	Channel Type
LongRetryInterval	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
SeqNumberWrap	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
MaxMsgLength	MQXR_INIT	All	All
PutAuthority	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
DataConversion	MQXR_INIT	All	All
MCAUserIdentifier	MQXR_INIT, MQXR_INIT_SEC, MQXR_SEC_MSG, MQXR_SEC_PARMS	Security	RCVR, RQSTR, SVRCONN, CLUSRCVR
ConnectionName	MQXR_INIT	All	SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, CLUSRCVR
MsgRetryUserData	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
MsgRetryCount	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
MsgRetryInterval	MQXR_INIT	All	RCVR, RQSTR, CLUSRCVR
HeartbeatInterval	MQXR_INIT	All	All
BatchInterval	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR
NonPersistentMsgSpeed	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
MCASecurityId	MQXR_INIT, MQXR_INIT_SEC, MQXR_SEC_MSG, MQXR_SEC_PARMS	Security	SDR, SVR, RCVR, RQSTR, SVRCONN, CLUSSDR, CLUSRCVR
SSLCipherSpec	MQXR_INIT	All	All

Field	Exit Reason Code	Exit Type	Channel Type
SSLPeerNamePtr	MQXR_INIT	All	All
SSLPeerNameLength	MQXR_INIT	All	All
SSLClientAuth	MQXR_INIT	All	SVR, RCVR, RQSTR, SVRCONN, CLUSRCVR
KeepAliveInterval	MQXR_INIT	All	All
LocalAddress	MQXR_INIT	All	SDR, SVR, RQSTR, CLNTCONN, CLUSSDR, CLUSRCVR
BatchHeartbeat	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR
HdrCompList	MQXR_INIT	All	All
MsgCompList	MQXR_INIT	All	All
ChannelMonitoring	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, SVRCONN, CLUSSDR, CLUSRCVR
ChannelStatistics	MQXR_INIT	All	SDR, SVR, RCVR, RQSTR, CLUSSDR, CLUSRCVR
SharingConversations	MQXR_INIT	All	SVRCONN, CLNTCONN
PropertyControl	MQXR_INIT	All	SDR, SVR, CLUSSDR, CLUSRCVR

MQCXP - Channel exit parameter:

The MQCXP structure is passed to each type of exit called by a Message Channel Agent (MCA), client-connection channel, or server-connection channel.

See MQ_CHANNEL_EXIT.

The fields described as "input to the exit" in the descriptions that follow are ignored by the channel when the exit returns control to the channel. Any input fields that the exit changes in the channel exit parameter block will not be preserved for its next invocation. Changes made to input/output fields (for example, the *ExitUserArea* field), are preserved for invocations of that instance of the exit only. Such changes cannot be used to pass data between different exits defined on the same channel, or between the same exit defined on different channels.

Related reference:

"Fields"
This topic lists all the fields in the MQCXP structure and describes each field.

"C declaration" on page 2407
This declaration is the C declaration for the MQCXP structure.

"COBOL declaration" on page 2408
This declaration is the COBOL declaration for the MQCXP structure.

"RPG declaration (ILE)" on page 2409
This declaration is the RPG declaration for the MQCXP structure.

"System/390 assembler declaration" on page 2410
This declaration is the System/390 assembler declaration for the MQCXP structure.

Fields:

This topic lists all the fields in the MQCXP structure and describes each field.

StrucId (MQCHAR4):

This field specifies the structure identifier.

The value must be:

MQCXP_STRUC_ID
Identifier for channel exit parameter structure.

For the C programming language, the constant MQCXP_STRUC_ID_ARRAY is also defined; this constant has the same value as MQCXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG):

This field specifies the structure version number.

The value depends on the environment:

MQCXP_VERSION_1
Version-1 channel exit parameter structure.

MQCXP_VERSION_2
Version-2 channel exit parameter structure.

The field has this value in the following environments: HP Integrity NonStop Server.

MQCXP_VERSION_3
Version-3 channel exit parameter structure.

The field has this value in the following environments: UNIX systems not listed elsewhere.

MQCXP_VERSION_4
Version-4 channel exit parameter structure.

MQCXP_VERSION_5
Version-5 channel exit parameter structure.

MQCXP_VERSION_6
Version-6 channel exit parameter structure.

MQCXP_VERSION_8
Version-8 channel exit parameter structure.

The field has this value in the following environments: z/OS, AIX, HP-UX, Linux, IBM i, Solaris, Windows.

Fields that exist only in the more-recent versions of the structure are identified as such in the descriptions of the fields. The following constant specifies the version number of the current version:

MQCXP_CURRENT_VERSION
Current version of channel exit parameter structure.

The value depends on the environment.

Note: When a new version of the MQCXP structure is introduced, the layout of the existing part is not changed. The exit must therefore check that the version number is equal to or greater than the lowest version which contains the fields that the exit needs to use.

This is an input field to the exit.

ExitId (MQLONG):

This field specifies the type of exit being called and is set on entry to the exit routine.

The following values are possible:

MQXT_CHANNEL_SEC_EXIT
Channel security exit.

MQXT_CHANNEL_MSG_EXIT
Channel message exit.

MQXT_CHANNEL_SEND_EXIT
Channel send exit.

MQXT_CHANNEL_RCV_EXIT
Channel receive exit.

MQXT_CHANNEL_MSG_RETRY_EXIT
Channel message-retry exit.

MQXT_CHANNEL_AUTO_DEF_EXIT
Channel auto-definition exit.

On z/OS, this type of exit is supported only for channels of type MQCHT_CLUSSDR and MQCHT_CLUSRCVR.

This is an input field to the exit.

ExitReason (MQLONG):

This field specifies the reason why the exit is being called and is set on entry to the exit routine.

It is not used by the auto-definition exit. The following values are possible:

MQXR_INIT
Exit initialization.

This value indicates that the exit is being invoked for the first time. It allows the exit to acquire and initialize any resources that it needs (for example: memory).

MQXR_TERM
Exit termination.

This value indicates that the exit is about to be terminated. The exit should free any resources that it has acquired since it was initialized (for example: memory).

MQXR_MSG
Process a message.

This value indicates that the exit is being invoked to process a message. This value occurs for channel message exits only.

MQXR_XMIT
Process a transmission.

This value occurs for channel send and receive exits only.

MQXR_SEC_MSG
Security message received.

This value occurs for channel security exits only.

MQXR_INIT_SEC
Initiate security exchange.

This value occurs for channel security exits only.

The security exit of the receiver is always invoked with this reason immediately after being invoked with MQXR_INIT, to give it the opportunity to initiate a security exchange. If it declines the opportunity (by returning MQXCC_OK instead of MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG), the security exit of the sender is invoked with MQXR_INIT_SEC.

If the security exit of the receiver does initiate a security exchange (by returning MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG), the security exit of the sender is never invoked with MQXR_INIT_SEC; instead it is invoked with MQXR_SEC_MSG to process the message of the receiver. (In either case it is first invoked with MQXR_INIT.)

Unless one of the security exits requests termination of the channel (by setting *ExitResponse* to MQXCC_SUPPRESS_FUNCTION or MQXCC_CLOSE_CHANNEL), the security exchange must complete at the side that initiated the exchange. Therefore, if a security exit is invoked with MQXR_INIT_SEC and it does initiate an exchange, the next time the exit is invoked it will be with MQXR_SEC_MSG. This happens whether there is a security message for the exit to process or not. There is a security message if the partner returns MQXCC_SEND_SEC_MSG or MQXCC_SEND_AND_REQUEST_SEC_MSG, but not if the partner returns MQXCC_OK or there is no security exit at the partner. If there is no security message to process, the security exit at the initiating end is re-invoked with a *DataLength* of zero.

MQXR_RETRY
Retry a message.

This value occurs for message-retry exits only.

MQXR_AUTO_CLUSSDR
Automatic definition of a cluster-sender channel.

This value occurs for channel auto-definition exits only.

MQXR_AUTO_RECEIVER
Automatic definition of a receiver channel.

This value occurs for channel auto-definition exits only.

MQXR_AUTO_SVRCONN
Automatic definition of a server-connection channel.

This value occurs for channel auto-definition exits only.

MQXR_AUTO_CLUSRCVR
Automatic definition of a cluster-receiver channel.

This value occurs for channel auto-definition exits only.

MQXR_SEC_PARMS
Security parameters

This value applies to security exits only and indicates that an MQCSP structure is being passed to the exit. For more information, see "MQCSP - Security parameters" on page 1621

Note:

1. If you have more than one exit defined for a channel, they are each invoked with MQXR_INIT when the MCA is initialized. Also, they are each invoked with MQXR_TERM when the MCA is terminated.
2. For the channel auto-definition exit, *ExitReason* is not set if *Version* is less than MQCXP_VERSION_4. The value MQXR_AUTO_SVRCONN is implied in this case.

This is an input field to the exit.

ExitResponse (MQLONG):

This field specifies the response from the exit.

This field is set by the exit to communicate with the MCA. It must be one of the following values:

MQXCC_OK
Exit completed successfully.

- For the channel security exit, this value indicates that message transfer can now proceed normally.
- For the channel message retry exit, this value indicates that the MCA must wait for the time interval returned by the exit in the *MsgRetryInterval* field in MQCXP, and then try the message again.

The *ExitResponse2* field might contain additional information.

MQXCC_SUPPRESS_FUNCTION
Suppress function.

- For the channel security exit, this value indicates that the channel must be terminated.
- For the channel message exit, this value indicates that the message is not to proceed any further towards its destination. Instead the MCA generates an exception report message (if one was requested by the sender of the original message), and places the message contained in the original buffer on the dead-letter queue (if the sender specified MQRO_DEAD_LETTER_Q), or discards it (if the sender specified MQRO_DISCARD_MSG).

  For persistent messages, if the sender specified MQRO_DEAD_LETTER_Q, but the put to the dead-letter queue fails, or there is no dead-letter queue, the original message is left on the transmission queue and the report message is not generated. The original message is also left on the transmission queue if the report message cannot be generated successfully.

  The *Feedback* field in the MQDLH structure at the start of the message on the dead-letter queue indicates why the message was put on the dead-letter queue; this feedback code is also used in the message descriptor of the exception report message (if one was requested by the sender).
- For the channel message retry exit, this value indicates that the MCA does not wait and try the message again; instead, the MCA continues immediately with its normal failure processing (the message is placed on the dead-letter queue or discarded, as specified by the sender of the message).

- For the channel auto-definition exit, either MQXCC_OK or MQXCC_SUPPRESS_FUNCTION must be specified. If neither of these values is specified, MQXCC_SUPPRESS_FUNCTION is assumed by default and the auto-definition is abandoned.

This response is not supported for the channel send and receive exits.

MQXCC_SEND_SEC_MSG
Send security message.

This value can be set only by a channel security exit. It indicates that the exit has provided a security message which must be transmitted to the partner.

MQXCC_SEND_AND_REQUEST_SEC_MSG
Send security message that requires a reply.

This value can be set only by a channel security exit. It indicates

- that the exit has provided a security message which can be transmitted to the partner, and
- that the exit requires a response from the partner. If no response is received, the channel must be terminated, because the exit has not yet decided whether communications can proceed.

MQXCC_SUPPRESS_EXIT
Suppress exit.

- This value can be set by all types of channel exit other than a security exit or an auto-definition exit. It suppresses any further invocation of that exit (as if its name had been blank in the channel definition), until termination of the channel, when the exit is again invoked with an *ExitReason* of MQXR_TERM.
- If a message retry exit returns this value, message retries for subsequent messages are controlled by the *MsgRetryCount* and *MsgRetryInterval* channel attributes as normal. For the current message, the MCA performs the number of outstanding retries, at intervals given by the *MsgRetryInterval* channel attribute, but only if the reason code is one that the MCA would normally retry (see the *MsgRetryCount* field described in "MQCD - Channel definition" on page 2348). The number of outstanding retries is the value of the *MsgRetryCount* attribute, less the number of times the exit returned MQXCC_OK for the current message; if this number is negative, no further retries are performed by the MCA for the current message.

MQXCC_CLOSE_CHANNEL
Close channel.

This value can be set by any type of channel exit except an auto-definition exit.

If sharing conversations is not enabled, this value closes the channel.

If sharing conversations is enabled, this value ends the conversation. If this conversation is the only conversation on the channel, the channel also closes.

This field is an input/output field from the exit.

ExitResponse2 (MQLONG):

This field specifies the secondary response from the exit.

This field is set to zero on entry to the exit routine. It can be set by the exit to provide further information to the WebSphere MQ channel functions. It is not used by the auto-definition exit.

The exit can set one or more of the following values. If more than one is required, the values are added. Combinations that are not valid are noted; other combinations are allowed.

MQXR2_PUT_WITH_DEF_ACTION
Put with default action.

This value is set by the channel message exit of the receiver. It indicates that the message is to be put with the default action of the MCA, that is either the default user ID of the MCA, or the context *UserIdentifier* in the MQMD (message descriptor) of the message.

The value is zero, which corresponds to the initial value set when the exit is invoked. The constant is provided for documentation purposes.

MQXR2_PUT_WITH_DEF_USERID
Put with default user identifier.

This value can only be set by the channel message exit of the receiver. It indicates that the message is to be put with the default user identifier of the MCA.

MQXR2_PUT_WITH_MSG_USERID
Put with user identifier of the message.

This value can only be set by the channel message exit of the receiver. It indicates that the message is to be put with the context *UserIdentifier* in the MQMD (message descriptor) of the message (this might have been modified by the exit).

Only one of MQXR2_PUT_WITH_DEF_ACTION, MQXR2_PUT_WITH_DEF_USERID, and MQXR2_PUT_WITH_MSG_USERID should be set.

MQXR2_USE_AGENT_BUFFER
Use agent buffer.

This value indicates that any data to be passed on is in *AgentBuffer*, not *ExitBufferAddr*.

The value is zero, which corresponds to the initial value set when the exit is invoked. The constant is provided for documentation purposes.

MQXR2_USE_EXIT_BUFFER
Use exit buffer.

This value indicates that any data to be passed on is in *ExitBufferAddr*, not *AgentBuffer*.

Only one of MQXR2_USE_AGENT_BUFFER and MQXR2_USE_EXIT_BUFFER should be set.

MQXR2_DEFAULT_CONTINUATION
Default continuation.

Continuation with the next exit in the chain depends on the response from the last exit invoked:

- If MQXCC_SUPPRESS_FUNCTION or MQXCC_CLOSE_CHANNEL are returned, no further exits in the chain are called.
- Otherwise, the next exit in the chain is invoked.

MQXR2_CONTINUE_CHAIN
Continue with the next exit.

MQXR2_SUPPRESS_CHAIN
Skip remaining exits in chain.

This is an input/output field to the exit.

Feedback (MQLONG):

This field specifies the feedback code.

This field is set to MQFB_NONE on entry to the exit routine.

If a channel message exit sets the *ExitResponse* field to MQXCC_SUPPRESS_FUNCTION, the *Feedback* field specifies the feedback code that identifies why the message was put on the dead-letter (undelivered-message) queue, and is also used to send an exception report if one has been requested. In this case, if the *Feedback* field is MQFB_NONE, the following feedback code is used:

MQFB_STOPPED_BY_MSG_EXIT
Message stopped by channel message exit.

The value returned in this field by channel security, send, receive, and message-retry exits is not used by the MCA.

The value returned in this field by auto-definition exits is not used if *ExitResponse* is MQXCC_OK, but otherwise is used for the *AuxErrorDataInt1* parameter in the event message.

This is an input/output field from the exit.

MaxSegmentLength (MQLONG):

This field specifies the maximum length in bytes that can be sent in a single transmission.

It is not used by the auto-definition exit. It is of interest to a channel send exit, because this exit must ensure that it does not increase the size of a transmission segment to a value greater than *MaxSegmentLength*. The length includes the initial 8 bytes that the exit must not change. The value is negotiated between the WebSphere MQ channel functions when the channel is initiated. See Writing channel-exit programs for more information about segment lengths.

The value in this field is not meaningful if *ExitReason* is MQXR_INIT.

This is an input field to the exit.

ExitUserArea (MQBYTE16):

This field specifies the exit user area - a field available for the exit to use.

It is initialized to binary zero before the first invocation of the exit (which has an *ExitReason* set to MQXR_INIT), and thereafter any changes made to this field by the exit are preserved across invocations of the exit.

The following value is defined:

MQXUA_NONE
No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this constant has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

ExitData (MQCHAR32):

This field specifies the exit data.

This field is set on entry to the exit routine to information that WebSphere MQ channel functions took from the channel definition. If no such information is available, this field is all blanks.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

This is an input field to the exit.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryCount (MQLONG):

This field specifies the number of times the message has been retried.

The first time the exit is invoked for a particular message, this field has the value zero (no retries yet attempted). On each subsequent invocation of the exit for that message, the value is incremented by one by the MCA.

This is an input field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryInterval (MQLONG):

This field specifies the minimum interval in milliseconds after which the put operation is retried.

The first time the exit is invoked for a particular message, this field contains the value of the *MsgRetryInterval* channel attribute. The exit can leave the value unchanged, or modify it to specify a different time interval in milliseconds. If the exit returns MQXCC_OK in *ExitResponse*, the MCA waits for at least this time interval before retrying the MQOPEN or MQPUT operation. The time interval specified must be zero or greater.

The second and subsequent times the exit is invoked for that message, this field contains the value returned by the previous invocation of the exit.

If the value returned in the *MsgRetryInterval* field is less than zero or greater than 999 999 999, and *ExitResponse* is MQXCC_OK, the MCA ignores the *MsgRetryInterval* field in MQCXP and waits instead for the interval specified by the *MsgRetryInterval* channel attribute.

This is an input/output field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

MsgRetryReason (MQLONG):

This field specifies the reason code from the previous attempt to put the message.

This field is the reason code from the previous attempt to put the message; it is one of the MQRC_* values.

This is an input field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_2.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_3.

HeaderLength (MQLONG):

This field specifies the length of header information.

This field is relevant only for a message exit and a message-retry exit. The value is the length of the routing header structures at the start of the message data; these are the MQXQH structure, the MQMDE (message description extension header), and (for a distribution-list message) the MQDH structure and arrays of MQOR and MQPMR records that follow the MQXQH structure.

The message exit can examine this header information, and modify it if necessary, but the data that the exit returns must still be in the correct format. The exit must not, for example, encrypt or compress the header data at the sending end, even if the message exit at the receiving end makes compensating changes.

If the message exit modifies the header information in such a way as to change its length (for example, by adding another destination to a distribution-list message), it must change the value of *HeaderLength* correspondingly before returning.

This is an input/output field to the exit. The value in this field is not meaningful if *ExitReason* is MQXR_INIT. The field is not present if *Version* is less than MQCXP_VERSION_3.

PartnerName (MQCHAR48):

This field specifies the name of the partner.

The name of the partner, as follows:

- For SVRCONN channels, it is the logged-on user ID at the client.
- For all other types of channel, it is the queue-manager name of the partner.

When the exit is initialized this field is blank because the queue manager does not know the name of the partner until after initial negotiation has taken place.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

FAPLevel (MQLONG):

Negotiated Formats and Protocols level.

This is an input field to the exit. Changes to this field should only be made under the direction of IBM service. The field is not present if *Version* is less than MQCXP_VERSION_3.

CapabilityFlags (MQLONG):

This field specifies the capability flags.

The following are defined:

MQCF_NONE
No flags.

MQCF_DIST_LISTS
Distribution lists supported.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

ExitNumber (MQLONG):

This field specifies the ordinal number of the exit.

The ordinal number of the exit, within the type defined in *ExitId*. For example, if the exit being invoked is the third message exit defined, this field contains the value 3. If the exit type is one for which a list of exits cannot be defined (for example, a security exit), this field has the value 1.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_3.

The following fields in this structure are not present if *Version* is less than MQCXP_VERSION_5.

ExitSpace (MQLONG):

This field specifies the number of bytes in the transmission buffer reserved for the exit to use.

This field is relevant only for a send exit. It specifies the amount of space in bytes that the WebSphere MQ channel functions reserve in the transmission buffer for the exit to use. This field allows the exit to add to the transmission buffer a small amount of data (typically not exceeding a few hundred bytes) for use by a complementary receive exit at the other end. The data added by the send exit must be removed by the receive exit.

The value is always zero on z/OS.

Note: This facility must not be used to send large amounts of data, as it might degrade performance, or even inhibit operation of the channel.

By setting *ExitSpace* the exit is guaranteed that there is always at least that number of bytes available in the transmission buffer for the exit to use. However, the exit can use less than the amount reserved, or more than the amount reserved if there is space available in the transmission buffer. The exit space in the buffer is provided following the existing data.

ExitSpace can be set by the exit only when *ExitReason* has the value MQXR_INIT; in all other cases the value returned by the exit is ignored. On input to the exit, *ExitSpace* is zero for the MQXR_INIT call, and is the value returned by the MQXR_INIT call in other cases.

If the value returned by the MQXR_INIT call is negative, or there are fewer than 1024 bytes available in the transmission buffer for message data after reserving the requested exit space for all the send exits in the chain, the MCA outputs an error message and closes the channel. Similarly, if during data transfer the exits in the send exit chain allocate more user space than they reserved such that fewer than 1024 bytes remain in the transmission buffer for message data, the MCA outputs an error message and closes the channel. The limit of 1024 allows the control and administrative flows of the channel to be processed by the chain of send exits, without the need for the flows to be segmented.

This is an input/output field to the exit if *ExitReason* is MQXR_INIT, and an input field in all other cases. The field is not present if *Version* is less than MQCXP_VERSION_5.

SSLCertUserId (MQCHAR12):

This field specifies the UserId associated with the remote certificate.

It is blank on all platforms except z/OS

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

SSLRemCertIssNameLength (MQLONG):

This field specifies the length in bytes of the full Distinguished Name of the issuer of the remote certificate pointed to by SSLCertRemoteIssuerNamePtr.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6. The value is zero if it is not an SSL channel.

SSLRemCertIssNamePtr (PMQVOID):

This field specifies the address of the full Distinguished Name of the issuer of the remote certificate.

Its value is the null pointer if it is not an SSL channel.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

Note: The behaviour of channel security exits in determining the Subject Distinguished Name and the Issuer Distinguished Name has altered in the release of WebSphere MQ v7.1. For more information see Channel security exit programs.

SecurityParms (PMQCSP):

This field specifies the address of the MQSCP structure used to specify a user ID and password.

The initial value of this field is the null pointer.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

CurHdrCompression (MQLONG):

This field specifies which technique is currently being used to compress the header data.

It is set to one of the following:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

The value can be altered by a sending channel's message exit to one of the negotiated supported values accessed from the HdrCompList field of the MQCD. This enables the technique used to compress the header data to be chosen for each message based on the content of the message. The altered value is used for the current message only. The channel ends if the attribute is altered to an unsupported value. The value is ignored if altered outside a sending channel's message exit.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

CurMsgCompression (MQLONG):

This field specifies which technique is currently being used to compress the message data.

It is set to one of the following:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using the zlib compression technique. A fast compression time is preferred.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using the zlib compression technique. A high level of compression is preferred.

The value can be altered by a sending channel's message exit to one of the negotiated supported values accessed from the MsgCompList field of the MQCD. This enables the technique used to compress the message data to be decided for each message based on the content of the message. The altered value is used for the current message only. The channel ends if the attribute is altered to an unsupported value. The value is ignored if altered outside a sending channel's message exit.

This is an input/output field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_6.

Hconn (MQHCONN):

This field specifies the connection handle that the exit uses if it needs to make any MQI calls within the exit.

This field is not relevant to exits running on client-connection channels, where it contains the value MQHC_UNUSABLE_HCONN (-1).

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_7.

SharingConversations (MQBOOL):

This field specifies whether the conversation is the only one that can currently be running on this channel instance, or whether more than one conversation can currently be running on this channel instance.

It also indicates whether the exit program is subject to the risk of the MQCD being altered by another exit program running at the same time.

This field is only relevant for exit programs running on client-connection or server-connection channels.

It is set to one of the following:

FALSE
The exit instance is the only exit instance that can currently be running on this channel instance. This allows the exit to safely update the MQCD fields without contention from other exits running on other channel instances. Whether changes to the MQCD fields are acted upon by the channel is defined by the table of MQCD fields in "Changing MQCD fields in a channel exit" on page 2392.

TRUE The exit instance is not the only exit instance that can currently be running on this channel

instance. Any changes made to the MQCD are not acted upon by the channel, except for changes listed in the table of MQCD fields in "Changing MQCD fields in a channel exit" on page 2392 for Exit Reasons other than MQXR_INIT. If this exit updates the MQCD fields, ensure there is no contention from other exits running on other conversations at the same time by providing serialization among the exits that run on this channel instance.

This is an input field to the exit. The field is not present if *Version* is less than MQCXP_VERSION_7.

MCAUserSource (MQLONG):

This field specifies the source of the provided MCA user ID.

It can contain one of the following values:

MQUSRC_MAP
The user ID is specified in the MCAUSER attribute.

MQUSRC_CHANNEL
The user ID is flowed from the inbound partner or specified in the MCAUSER field defined in the channel object.

This is an input field to the exit. The field is not present if Version is less than MQCXP_VERSION_8.

pEntryPoints (PMQIEP):

This field specifies the address of the interface entry point for the MQI or DCI call.

The field is not present if *Version* is less than MQCXP_VERSION_8.

C declaration:

This declaration is the C declaration for the MQCXP structure.

```
typedef struct tagMQCXP MQCXP;
struct tagMQCXP {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    ExitId;            /* Type of exit */
  MQLONG    ExitReason;        /* Reason for invoking exit */
  MQLONG    ExitResponse;      /* Response from exit */
  MQLONG    ExitResponse2;     /* Secondary response from exit */
  MQLONG    Feedback;          /* Feedback code */
  MQLONG    MaxSegmentLength;  /* Maximum segment length */
  MQBYTE16  ExitUserArea;      /* Exit user area */
  MQCHAR32  ExitData;          /* Exit data */
  MQLONG    MsgRetryCount;     /* Number of times the message has been
                                  retried */
  MQLONG    MsgRetryInterval;  /* Minimum interval in milliseconds after
                                  which the put operation should be
                                  retried */
  MQLONG    MsgRetryReason;    /* Reason code from previous attempt to
                                  put the message */
  MQLONG    HeaderLength;      /* Length of header information */
  MQCHAR48  PartnerName;       /* Partner Name */
  MQLONG    FAPLevel;          /* Negotiated Formats and Protocols
                                  level */
  MQLONG    CapabilityFlags;   /* Capability flags */
  MQLONG    ExitNumber;        /* Exit number */
  /* Ver:3 */
  /* Ver:4 */
  MQLONG    ExitSpace;         /* Number of bytes in transmission buffer
                                  reserved for exit to use */
  /* Ver:5 */
```

```
  MQCHAR12  SSLCertUserid;     /* User identifier associated
                                  with remote SSL certificate */
  MQLONG    SSLRemCertIssNameLength;   /* Length of
                                  distinguished name of issuer
                                  of remote SSL certificate */
  MQPTR     SSLRemCertIssNamePtr;      /* Address of
                                  distinguished name of issuer
                                  of remote SSL certificate */
  PMQVOID   SecurityParms;     /* Security parameters */
  MQLONG    CurHdrCompression; /* Header data compression
                                  used for current message */
  MQLONG    CurMsgCompression; /* Message data compression
                                  used for current message */
  /* Ver:6 */
  MQHCONN   Hconn;             /* Connection handle */
  MQBOOL    SharingConversations;      /* Multiple conversations
                                  possible on channel inst?  */
  /* Ver:7 */
  MQLONG    MCAUserSource;     /* Source of the provided MCA user ID */
  PMQIEP    pEntryPoints;     /* Address of the MQIEP structure */
}
  /* Ver:8 */
;
```

COBOL declaration:

This declaration is the COBOL declaration for the MQCXP structure.

```
**   MQCXP structure
  10 MQCXP.
**    Structure identifier
   15 MQCXP-STRUCID          PIC X(4).
**    Structure version number
   15 MQCXP-VERSION          PIC S9(9) BINARY.
**    Type of exit
   15 MQCXP-EXITID           PIC S9(9) BINARY.
**    Reason for invoking exit
   15 MQCXP-EXITREASON       PIC S9(9) BINARY.
**    Response from exit
   15 MQCXP-EXITRESPONSE     PIC S9(9) BINARY.
**    Secondary response from exit
   15 MQCXP-EXITRESPONSE2    PIC S9(9) BINARY.
**    Feedback code
   15 MQCXP-FEEDBACK         PIC S9(9) BINARY.
**    Maximum segment length
   15 MQCXP-MAXSEGMENTLENGTH PIC S9(9) BINARY.
**    Exit user area
   15 MQCXP-EXITUSERAREA     PIC X(16).
**    Exit data
   15 MQCXP-EXITDATA         PIC X(32).
**    Number of times the message has been retried
   15 MQCXP-MSGRETRYCOUNT    PIC S9(9) BINARY.
**    Minimum interval in milliseconds after which the put operation
**    should be retried
   15 MQCXP-MSGRETRYINTERVAL PIC S9(9) BINARY.
**    Reason code from previous attempt to put the message
   15 MQCXP-MSGRETRYREASON   PIC S9(9) BINARY.
**    Length of header information
   15 MQCXP-HEADERLENGTH     PIC S9(9) BINARY.
**    Partner Name
   15 MQCXP-PARTNERNAME      PIC X(48).
**    Negotiated Formats and Protocols level
   15 MQCXP-FAPLEVEL         PIC S9(9) BINARY.
**    Capability flags
   15 MQCXP-CAPABILITYFLAGS  PIC S9(9) BINARY.
**    Exit number
   15 MQCXP-EXITNUMBER       PIC S9(9) BINARY.
```

```
**    Number of bytes in transmission buffer reserved for exit to use
   15 MQCXP-EXITSPACE        PIC S9(9) BINARY.
**    User Id associated with remote certificate
   15  MQCXP-SSLCERTUSERID   PIC X(12).
** Length of distinguished name of issuer of remote SSL
** certificate
   15  MQCXP-SSLREMCERTISSNAMELENGTH PIC S9(9) BINARY.
** Address of distinguished name of issuer of remote SSL
** certificate
   15  MQCXP-SSLREMCERTISSNAMEPTR    POINTER.
** Security parameters
   15  MQCXP-SECURITYPARMS           PIC S9(18) BINARY.
** Header data compression used for current message
   15  MQCXP-CURHDRCOMPRESSION       PIC S9(9) BINARY.
** Message data compression used for current message
   15  MQCXP-CURMSGCOMPRESSION       PIC S9(9) BINARY.
** Connection handle
   15 MQCXP-HCONN                    PIC S9(9) BINARY.
** Multiple conversations possible on channel instance?
   15 MQCXP-SHARINGCONVERSATIONS     PIC S9(9) BINARY.
** Source of the provided MCA user ID
   15 MQCXP-MCAUSERSOURCE            PIC S9(9) BINARY.
```

RPG declaration (ILE):

This declaration is the RPG declaration for the MQCXP structure.

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCXP Structure
     D*
     D* Structure identifier
     D  CXSID                  1      4
     D* Structure version number
     D  CXVER                  5      8I 0
     D* Type of exit
     D  CXXID                  9     12I 0
     D* Reason for invoking exit
     D  CXREA                 13     16I 0
     D* Response from exit
     D  CXRES                 17     20I 0
     D* Secondary response from exit
     D  CXRE2                 21     24I 0
     D* Feedback code
     D  CXFB                  25     28I 0
     D* Maximum segment length
     D  CXMSL                 29     32I 0
     D* Exit user area
     D  CXUA                  33     48
     D* Exit data
     D  CXDAT                 49     80
     D* Number of times the message has been retried
     D  CXMRC                 81     84I 0
     D* Minimum interval in milliseconds after which the put operation
     D* should be retried
     D  CXMRI                 85     88I 0
     D* Reason code from previous attempt to put the message
     D  CXMRR                 89     92I 0
     D* Length of header information
     D  CXHDL                 93     96I 0
     D* Partner Name
     D  CXPNM                 97    144
     D* Negotiated Formats and Protocols level
     D  CXFAP                145    148I 0
     D* Capability flags
     D  CXCAP                149    152I 0
     D* Exit number
     D  CXEXN                153    156I 0
```

```
     D* Number of bytes in transmission buffer reserved for exit to use
     D  CXHDL                157    160I 0
     D* User identifier associated with remote SSL certificate
     D  CXSSLCU              161    172
     D* Length of distinguished name of issuer of remote SSL certificate
     D  CXSRCINL             173    176I 0
     D* Address of distinguished name of issuer of remote SSL certificate
     D  CXSRCINP             177    192*
     D* Security parameters
     D  CXSECP               193    208*
     D* Header data compression used for current message
     D  CXCHC                209    212I 0
     D* Message data compression used for current message
     D  CXCMC                213    216I 0
     D* Connection handle
     D  CXHCONN               217    220I   0
     D* Multiple conversations possible on channel instance?
     D  CXSHARECONV           221    224I   0
     D* Source of the provided MCA user ID
     D  MCAUSERSOURCE         225    228I   0
```

System/390 assembler declaration:

This declaration is the System/390 assembler declaration for the MQCXP structure.

```
MQCXP                   DSECT
MQCXP_STRUCID           DS   CL4   Structure identifier
MQCXP_VERSION           DS   F     Structure version number
MQCXP_EXITID            DS   F     Type of exit
MQCXP_EXITREASON        DS   F     Reason for invoking exit
MQCXP_EXITRESPONSE      DS   F     Response from exit
MQCXP_EXITRESPONSE2     DS   F     Secondary response from exit
MQCXP_FEEDBACK          DS   F     Feedback code
MQCXP_MAXSEGMENTLENGTH  DS   F     Maximum segment length
MQCXP_EXITUSERAREA      DS   XL16  Exit user area
MQCXP_EXITDATA          DS   CL32  Exit data
MQCXP_MSGRETRYCOUNT     DS   F     Number of times the message has been
*                                  retried
MQCXP_MSGRETRYINTERVAL  DS   F     Minimum interval in milliseconds
*                                  after which the put operation should
*                                  be retried
MQCXP_MSGRETRYREASON    DS   F     Reason code from previous attempt to
*                                  put the message
MQCXP_HEADERLENGTH      DS   F     Length of header information
MQCXP_PARTNERNAME       DS   CL48  Partner Name
MQCXP_FAPLEVEL          DS   F     Negotiated Formats and Protocols
*                                  level
MQCXP_CAPABILITYFLAGS   DS   F     Capability flags
MQCXP_EXITNUMBER        DS   F     Exit number
MQCXP_EXITSPACE         DS   F     Number of bytes in transmission
*                                  buffer reserved for exit to use
MQCXP_SSLCERTUSERID     DS   CL12  User identifier associated with
*                                  remote SSL certificate
MQCXP_SSLREMCERTISSNAMELENGTH   DS   F     Length of distinguished name
*                                  of issuer of remote SSL certificate
MQCXP_SSLREMCERTISSNAMEPTR      DS   F     Address of distinguished name
*                                  of issuer of remote SSL certificate
MQCXP_SECURITYPARMS     DS   F    Address of security parameters
MQCXP_CURHDRCOMPRESSION         DS   F     Header data compression used for
*                                  current message
MQCXP_CURMSGCOMPRESSION         DS   F      Message data compression used for
*                                  current message
MQCXP_HCONN             DS   F Connection handle
MQCXP_SHARINGCONVERSATIONS      DS   F Multiple conversations possible on
*                                  channel inst?
MQCXP_MCAUSERSOURCE     DS   F    Source of the provided MCA user ID
```

```
MQCXP_LENGTH            EQU  *-MQCXP
                        ORG  MQCXP
MQCXP_AREA              DS   CL(MQCXP_LENGTH)
```

MQXWD - Exit wait descriptor:

The MQXWD structure is an input/output parameter on the MQXWAIT call.

This structure is supported only on z/OS.

Related reference:

"Fields"
This topic lists all the fields in the MQXWD structure and describes each field.

"C declaration" on page 2412
This declaration is the C declaration for the MQXWD structure.

"System/390 assembler declaration" on page 2412
This declaration is the System/390 assembler declaration for the MQXWD structure.

Fields:

This topic lists all the fields in the MQXWD structure and describes each field.

StrucId (MQCHAR4):

This field specifies the structure identifier.

The value must be:

MQXWD_STRUC_ID
Identifier for exit wait descriptor structure.

For the C programming language, the constant MQXWD_STRUC_ID_ARRAY is also defined; this constant has the same value as MQXWD_STRUC_ID, but is an array of characters instead of a string.

The initial value of this field is MQXWD_STRUC_ID.

Version (MQLONG):

This field specifies the structure version number.

The value must be:

MQXWD_VERSION_1
Version number for exit wait descriptor structure.

The initial value of this field is MQXWD_VERSION_1.

Reserved1 (MQLONG):

This field is reserved. Its value must be zero.

This is an input field.

Reserved2 (MQLONG):

This field is reserved. Its value must be zero.

This is an input field.

Reserved3 (MQLONG):

This field is reserved. Its value must be zero.

This is an input field.

ECB (MQLONG):

This field specifies the event control block to wait on.

This field is the event control block (ECB) to wait on. It must be set to zero before the MQXWAIT call is issued; on successful completion it contains the post code.

This field is an input/output field.

C declaration:

This declaration is the C declaration for the MQXWD structure.

```
typedef struct tagMQXWD MQXWD;
struct tagMQXWD {
  MQCHAR4  StrucId;    /* Structure identifier */
  MQLONG   Version;    /* Structure version number */
  MQLONG   Reserved1;  /* Reserved */
  MQLONG   Reserved2;  /* Reserved */
  MQLONG   Reserved3;  /* Reserved */
  MQLONG   ECB;        /* Event control block to wait on */
};
```

System/390 assembler declaration:

This declaration is the System/390 assembler declaration for the MQXWD structure.

```
MQXWD            DSECT
MQXWD_STRUCID    DS   CL4  Structure identifier
MQXWD_VERSION    DS   F    Structure version number
MQXWD_RESERVED1  DS   F    Reserved
MQXWD_RESERVED2  DS   F    Reserved
MQXWD_RESERVED3  DS   F    Reserved
MQXWD_ECB        DS   F    Event control block to wait on
*
MQXWD_LENGTH     EQU  *-MQXWD
                 ORG  MQXWD
MQXWD_AREA       DS   CL(MQXWD_LENGTH)
```

API exit reference

This section provides reference information mainly of interest to a programmer writing API exits.

General usage notes

notes:

1. All exit functions can issue the MQXEP call; this call is designed specifically for use from API exit functions.
2. The MQ_INIT_EXIT function cannot issue any MQ calls other than MQXEP.
3. You cannot issue the MQDISC call for the current connection.
4. If an exit function issues the MQCONN call, or the MQCONNX call with the MQCNO_HANDLE_SHARE_NONE option, the call completes with reason code MQRC_ALREADY_CONNECTED, and the handle returned is the same as the one passed to the exit as a parameter.
5. In general when an API exit function issues an MQI call, API exits are not be called recursively. However, if an exit function issues the MQCONNX call with the MQCNO_HANDLE_SHARE_BLOCK or MQCNO_HANDLE_SHARE_NO_BLOCK options, the call returns a new shared handle. This provides the exit suite with a connection handle of its own, and hence a unit of work that is independent of the application's unit of work. The exit suite can use this handle to put and get messages within its own unit of work, and commit or back out that unit of work; all of this can be done without affecting the application's unit of work in any way.

   Because the exit function is using a connection handle that is different from the handle being used by the application, MQ calls issued by the exit function result in the relevant API exit functions being invoked. Exit functions can therefore be invoked recursively. Note that both the *ExitUserArea* field in MQAXP and the exit chain area have connection-handle scope. Consequently, an exit function cannot use those areas to signal to another instance of itself invoked recursively that it is already active.
6. Exit functions can also put and get messages within the application's unit of work. When the application commits or backs out the unit of work, all messages within the unit of work are committed or backed out together, regardless of who placed them in the unit of work (application or exit function). However, the exit can cause the application to exceed system limits sooner than would otherwise be the case (for example, by exceeding the maximum number of uncommitted messages in a unit of work).

   When an exit function uses the application's unit of work in this way, the exit function should usually avoid issuing the MQCMIT call, as this commits the application's unit of work and might impair the correct functioning of the application. However, the exit function might sometimes need to issue the MQBACK call, if the exit function encounters a serious error that prevents the unit of work being committed (for example, an error putting a message as part of the application's unit of work). When MQBACK is called, take care to ensure that the application unit of work boundaries are not changed. In this situation the exit function must set the appropriate values to ensure that completion code MQCC_WARNING and reason code MQRC_BACKED_OUT are returned to the application, so that the application can detect the fact that the unit of work has been backed out.

   If an exit function uses the application's connection handle to issue MQ calls, those calls do not themselves result in further invocations of API exit functions.
7. If an MQXR_BEFORE exit function terminates abnormally, the queue manager might be able to recover from the failure. If it can, the queue manager continues processing as though the exit function had returned MQXCC_FAILED. If the queue manager cannot recover, the application is terminated.
8. If an MQXR_AFTER exit function terminates abnormally, the queue manager might be able to recover from the failure. If it can, the queue manager continues processing as though the exit function had returned MQXCC_FAILED. If the queue manager cannot recover, the application is

terminated. Be aware that in the latter case, messages retrieved outside a unit of work are lost (this is the same situation as the application failing immediately after removing a message from the queue).

9. The MCA process performs a two phase commit.

   If an API exit intercepts an MQCMIT from a prepared MCA process and attempts to perform an action within the unit of work, then the action will fail with reason code MQRC_UOW_NOT_AVAILABLE.

10. For a multi-installation environment, the only way to have an exit that works with both Websphere MQ version 7.0 and version 7.1 is to write the exit in a way which links at version 7.0 with mqm.Lib and, for non-primary or relocated exits, to ensure that the application finds the correct mqm.Lib for the installation with which the queue manager is currently associated, prior to the application launch. (For example, run the **setmqenv -m QM** command before launching the application, even if the queue manager is owned by a version 7.0 installation.)

11. Where multiple installations of WebSphere MQ are available, use the exits written for an earlier version of WebSphere MQ, as new functionality added in the later version might not work with earlier versions. For more information about changes between releases, see What's changed in WebSphere MQ 7.5.

WebSphere MQ API exit parameter structure (MQAXP):

The MQAXP structure, an external control block, is used as an input or output parameter to the API exit. This topic also gives information about how queue managers process exit functions.

MQAXP has the following C declaration:

```
typedef struct tagMQAXP {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    ExitId;            /* Exit Identifier */
  MQLONG    ExitReason;        /* Exit invocation reason */
  MQLONG    ExitResponse;      /* Response code from exit */
  MQLONG    ExitResponse2;     /* Secondary response code from exit */
  MQLONG    Feedback;          /* Feedback code from exit */
  MQLONG    APICallerType;     /* MQSeries API caller type */
  MQBYTE16  ExitUserArea;      /* User area for use by exit */
  MQCHAR32  ExitData;          /* Exit data area */
  MQCHAR48  ExitInfoName;      /* Exit information name */
  MQBYTE48  ExitPDArea;        /* Problem determination area */
  MQCHAR48  QMgrName;          /* Name of local queue manager */
  PMQACH    ExitChainAreaPtr;  /* Inter exit communication area */
  MQHCONFIG Hconfig;           /* Configuration handle */
  MQLONG    Function;          /* Function Identifier */
  /* Ver:1 */
  MQHMSG    ExitMsgHandle      /* Exit message handle
  /* Ver:2 */
};
```

The following parameter list is passed when functions in an API exit are invoked:

StrucId (MQCHAR4) - input
The exit parameter structure identifier, with a value of:

`MQAXP_STRUC_ID.`

The exit handler sets this field on entry to each exit function.

Version (MQLONG) - input
The structure version number, with a value of:

MQAXP_VERSION_1
Version 1 API exit parameter structure.

MQAXP_VERSION_2
Version 2 API exit parameter structure.

MQAXP_CURRENT_VERSION
Current version number for the API exit parameter structure.

The exit handler sets this field on entry to each exit function.

ExitId (MQLONG) - input
The exit identifier, set on entry to the exit routine, indicating the type of exit:

MQXT_API_EXIT
API exit.

ExitReason (MQLONG) - input
The reason for invoking the exit, set on entry to each exit function:

MQXR_CONNECTION
The exit is being invoked to initialize itself before an MQCONN or MQCONNX call, or to end itself after an MQDISC call.

MQXR_BEFORE
The exit is being invoked before executing an API call, or before converting data on an MQGET.

MQXR_AFTER
The exit is being invoked after executing an API call.

ExitResponse (MQLONG) - output
The response from the exit, initialized on entry to each exit function to:

MQXCC_OK
Continue normally.

This field must be set by the exit function, to communicate to the queue manager the result of executing the exit function. The value must be one of the following:

MQXCC_OK
The exit function completed successfully. Continue normally.

This value can be set by all MQXR_* exit functions. ExitResponse2 is used to decide whether to invoke exit functions later in the chain.

MQXCC_FAILED
The exit function failed because of an error.

This value can be set by all MQXR_* exit functions. The queue manager sets CompCode to MQCC_FAILED, and Reason to:

- MQRC_API_EXIT_INIT_ERROR if the function is MQ_INIT_EXIT
- MQRC_API_EXIT_TERM_ERROR if the function is MQ_TERM_EXIT
- MQRC_API_EXIT_ERROR for all other exit functions

The values set can be altered by an exit function later in the chain.

ExitResponse2 is ignored; the queue manager continues processing as though MQXR2_SUPPRESS_CHAIN had been returned.

MQXCC_SUPPRESS_FUNCTION
Suppress WebSphere MQ API function.

This value can be set only by an MQXR_BEFORE exit function. It bypasses the API call. If it is returned by the MQ_DATA_CONV_ON_GET_EXIT, data conversion is bypassed. The queue manager sets CompCode to MQCC_FAILED, and Reason to MQRC_SUPPRESSED_BY_EXIT, but the values set can be altered by an exit function later

in the chain. Other parameters for the call remain as the exit left them. ExitResponse2 is used to decide whether to invoke exit functions later in the chain.

If this value is set by an MQXR_AFTER or MQXR_CONNECTION exit function, the queue manager continues processing as though MQXCC_FAILED had been returned.

MQXCC_SKIP_FUNCTION
Skip WebSphere MQ API function.

This value can be set only by an MQXR_BEFORE exit function. It bypasses the API call. If it is returned by the MQ_DATA_CONV_ON_GET_EXIT, data conversion is bypassed. The exit function must set CompCode and Reason to the values to be returned to the application, but the values set can be altered by an exit function later in the chain. Other parameters for the call remain as the exit left them. ExitResponse2 is used to decide whether to invoke exit functions later in the chain.

If this value is set by an MQXR_AFTER or MQXR_CONNECTION exit function, the queue manager continues processing as though MQXCC_FAILED had been returned.

MQXCC_SUPPRESS_EXIT
Suppress all exit functions belonging to the set of exits.

This value can be set only by the MQXR_BEFORE and MQXR_AFTER exit functions. It bypasses *all* subsequent invocations of exit functions belonging to this set of exits for this logical connection. This bypassing continues until the logical disconnect request occurs, when MQ_TERM_EXIT function is invoked with an ExitReason of MQXR_CONNECTION.

The exit function must set CompCode and Reason to the values to be returned to the application, but the values set can be altered by an exit function later in the chain. Other parameters for the call remain as the exit left them. ExitResponse2 is ignored.

If this value is set by an MQXR_CONNECTION exit function, the queue manager continues processing as though MQXCC_FAILED had been returned.

For information about the interaction between ExitResponse and ExitResponse2, and its effect on exit processing, see "How queue managers process exit functions" on page 2418.

ExitResponse2 (MQLONG) - output
This is a secondary exit response code that qualifies the primary exit response code for MQXR_BEFORE exit functions. It is initialized to:

```
MQXR2_DEFAULT_CONTINUATION
```

on entry to a WebSphere MQ API call exit function. It can then be set to one of the values:

MQXR2_DEFAULT_CONTINUATION
Whether to continue with the next exit in the chain, depending on the value of ExitResponse.

If ExitResponse is MQXCC_SUPPRESS_FUNCTION or MQXCC_SKIP_FUNCTION, bypass exit functions later in the MQXR_BEFORE chain and the matching exit functions in the MQXR_AFTER chain. Invoke exit functions in the MQXR_AFTER chain that match exit functions earlier in the MQXR_BEFORE chain.

Otherwise, invoke the next exit in the chain.

MQXR2_SUPPRESS_CHAIN
Suppress the chain.

Bypass exit functions later in the MQXR_BEFORE chain and the matching exit functions in the MQXR_AFTER chain for this API call invocation. Invoke exit functions in the MQXR_AFTER chain that match exit functions earlier in the MQXR_BEFORE chain.

MQXR2_CONTINUE_CHAIN
Continue with the next exit in the chain.

For information about the interaction between ExitResponse and ExitResponse2, and its effect on exit processing, see "How queue managers process exit functions" on page 2418.

Feedback (MQLONG) - input/output
Communicate feedback codes between exit function invocations. This is initialized to:

```
MQFB_NONE (0)
```

before invoking the first function of the first exit in a chain.

Exits can set this field to any value, including any valid MQFB_* or MQRC_* value. Exits can also set this field to a user-defined feedback value in the range MQFB_APPL_FIRST to MQFB_APPL_LAST.

APICallerType (MQLONG) - input
The API caller type, indicating whether the WebSphere MQ API caller is external or internal to the queue manager: MQXACT_EXTERNAL or MQXACT_INTERNAL.

ExitUserArea (MQBYTE16) - input/output
A user area, available to all the exits associated with a particular ExitInfoObject. It is initialized to MQXUA_NONE (binary zeros for the length of the ExitUserArea) before invoking the first exit function (MQ_INIT_EXIT) for the hconn. From then on, any changes made to this field by an exit function are preserved across invocations of functions of the same exit.

This field is aligned to a multiple of 4 MQLONGs.

Exits can also anchor any storage that they allocate from this area.

For each hconn, each exit in a chain of exits has a different ExitUserArea. The ExitUserArea cannot be shared by exits in a chain, and the contents of the ExitUserArea for one exit are not available to another exit in a chain.

For C programs, the constant MQXUA_NONE_ARRAY is also defined with the same value as MQXUA_NONE, but as an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH.

ExitData (MQCHAR32) - input
Exit data, set on input to each exit function to the 32 characters of exit-specific data that is provided in the exit. If you define no value in the exit this field is all blanks.

The length of this field is given by MQ_EXIT_DATA_LENGTH.

ExitInfoName (MQCHAR48) - input
The exit information name, set on input to each exit function to the ApiExit_name specified in the exit definitions in the stanzas.

ExitPDArea (MQBYTE48) - input/output
A problem determination area, initialized to `MQXPDA_NONE` (binary zeros for the length of the field) for each invocation of an exit function.

For C programs, the constant MQXPDA_NONE_ARRAY is also defined with the same value as MQXPDA_NONE, but as an array of characters instead of a string.

The exit handler always writes this area to the WebSphere MQ trace at the end of an exit, even when the function is successful.

The length of this field is given by MQ_EXIT_PD_AREA_LENGTH.

QMgrName (MQCHAR48) - input
The name of the queue manager the application is connected to, that has invoked an exit as a result of processing a WebSphere MQ API call.

If the name of a queue manager supplied on an MQCONN or MQCONNX calls is blank, this field is still set to the name of the queue manager to which the application is connected, whether the application is server or client.

The exit handler sets this field on entry to each exit function.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH.

ExitChainAreaPtr (PMQACH) - input/output
This is used to communicate data across invocations of different exits in a chain. It is set to a NULL pointer before invoking the first function (MQ_INIT_EXIT with ExitReason MQXR_CONNECTION) of the first exit in a chain of exits. The value returned by the exit on one invocation is passed on to the next invocation.

Refer to "The exit chain area and exit chain area header (MQACH)" on page 2422 for more details about how to use the exit chain area.

Hconfig (MQHCONFIG) - input
The configuration handle, representing the set of functions being initialized. This value is generated by the queue manager on the MQ_INIT_EXIT function, and is later passed to the API exit function. It is set on entry to each exit function.

You can use Hconfig as a pointer to the MQIEP structure to make MQI and DCI calls. You must check that the first 4 bytes of HConfig match the StrucId of the MQIEP structure before using the HConfig parameter as a pointer to the MQIEP structure.

Function (MQLONG) - input
The function identifier, valid values for which are the MQXF_* constants described in "External constants" on page 2424.

The exit handler sets this field to the correct value, on entry to each exit function, depending on the WebSphere MQ API call that resulted in the exit being invoked.

ExitMsgHandle (MQHMSG) - input/output
When Function is MQXF_GET and ExitReason is MQXR_AFTER, a valid message handle is returned in this field allowing the API exit access to the message descriptor fields and any other properties matching the ExitProperties string specified in the MQXEPO structure when registering the API exit.

Any non-message descriptor properties that are returned in the ExitMsgHandle will not be available from the MsgHandle in the MQGMO structure if one was specified, or in the message data.

When Function is MQXF_GET and ExitReason is MQXR_BEFORE, if the exit program sets this field to MQHM_NONE then it will suppress the populating of the ExitMsgHandle properties.

This field is not set if Version is less then MQAXP_VERSION_2.

How queue managers process exit functions

The processing performed by the queue manager on return from an exit function depends on both ExitResponse and ExitResponse2.

Table 239 on page 2419 summarizes the possible combinations and their effects for an MQXR_BEFORE exit function, showing:

- Who sets the CompCode and Reason parameters of the API call
- Whether the remaining exit functions in the MQXR_BEFORE chain and the matching exit functions in the MQXR_AFTER chain are invoked
- Whether the API call is invoked

For an MQXR_AFTER exit function:

- CompCode and Reason are set in the same way as MQXR_BEFORE
- ExitResponse2 is ignored (the remaining exit functions in the MQXR_AFTER chain are always invoked)
- MQXCC_SUPPRESS_FUNCTION and MQXCC_SKIP_FUNCTION are not valid

For an MQXR_CONNECTION exit function:

- CompCode and Reason are set in the same way as MQXR_BEFORE
- ExitResponse2 is ignored
- MQXCC_SUPPRESS_FUNCTION, MQXCC_SKIP_FUNCTION, MQXCC_SUPPRESS_EXIT are not valid

In all cases, where an exit or the queue manager sets CompCode and Reason, the values set can be changed by an exit invoked later, or by the API call (if the API call is invoked later).

Table 239. MQXR_BEFORE exit processing

Value of ExitResponse	CompCode and Reason set by	Value of ExitResponse2 (default continuation) Chain	Value of ExitResponse2 (default continuation) API
MQXCC_OK	exit	Y	Y
MQXCC_SUPPRESS_EXIT	exit	Y	Y
MQXCC_SUPPRESS_FUNCTION	queue manager	N	N
MQXCC_SKIP FUNCTION	exit	N	N
MQXCC_FAILED	queue manager	N	N

How clients process exit functions

In general, clients process exit functions in the same way that server applications do, and the *QMgrName* attribute in this structure applies whether the function is on a server or a client.

However, the client has no concept of the *mqs.ini* file, so the *ApiExitCommon* and *APIExitTemplate* stanzas do not apply. Only the *ApiExitLocal* stanza applies, and this stanza is configured in the *mqclient.ini* file.

WebSphere MQ API exit context structure (MQAXC):

The MQAXC structure, an external control block, is used as an input parameter to an API exit.

MQAXC has the following C declaration:

```
typedef struct tagMQAXC {
   MQCHAR4   StrucId;                /* Structure identifier */
   MQLONG    Version;                /* Structure version number */
   MQLONG    Environment;            /* Environment */
   MQCHAR12  UserId;                 /* UserId associated with appl */
   MQBYTE40  SecurityId              /* Extension to UserId running appl */
   MQCHAR264 ConnectionName;         /* Connection name */
   MQLONG    LongMCAUserIdLength;    /* long MCA user identifier length */
   MQLONG    LongRemoteUserIdLength; /* long remote user identifier length */
   MQPTR     LongMCAUserIdPtr;       /* long MCA user identifier address */
   MQPTR     LongRemoteUserIdPtr;    /* long remote user identifier address */
   MQCHAR28  ApplName;               /* Application name */
   MQLONG    ApplType;               /* Application type */
   MQPID     ProcessId;              /* Process identifier */
   MQTID     ThreadId;               /* Thread identifier */

   /* Ver:1 */
   MQCHAR    ChannelName[20]         /* Channel Name */
   MQBYTE4   Reserved1;              /* Reserved */
   PMQCD     pChannelDefinition;     /* Channel Definition pointer */
 };
```

The parameters to MQAXC are:

StrucId (MQCHAR4) - input
The exit context structure identifier, with a value of `MQAXC_STRUC_ID`. For C programs, the constant MQAXC_STRUC_ID_ARRAY is also defined, with the same value as MQAXC_STRUC_ID, but as an array of characters instead of a string.

The exit handler sets this field on entry to each exit function.

Version (MQLONG) - input
The structure version number, with a value of:

MQAXC_VERSION_2
Version number for the exit context structure.

MQAXC_CURRENT_VERSION
Current version number for the exit context structure.

The exit handler sets this field on entry to each exit function.

Environment (MQLONG) - input
The environment from which a WebSphere MQ API call was issued that resulted in an exit function being driven. Valid values for this field are:

MQXE_OTHER
This value is consistent with invocations an API exit sees if the exit is called from a server application. This means that an API exit runs unchanged on a client and does not see anything different.

If the exit really needs to determine whether it is running on the client, the exit can do so by looking at the *ChannelName* and *ChannelDefinition* fields.

MQXE_MCA
Message channel agent

MQXE_MCA_SVRCONN
A message channel agent acting on behalf of a client

MQXE_COMMAND_SERVER
The command server

MQXE_MQSC
The runmqsc command interpreter

The exit handler sets this field on entry to each exit function.

UserId (MQCHAR12) - input
The user ID associated with the application. In particular, in the case of client connections, this field contains the user ID of the adopted user as opposed to the user ID under which the channel code is running. If a blank user ID flows from the client, then no change is made to the user ID already being used. That is, no new user ID is adopted.

The exit handler sets this field on entry to each exit function. The length of this field is given by MQ_USER_ID_LENGTH.

In the case of a client, this is the user ID sent from the client to the server. Note, that this might not be the effective user ID the client is running against in the queue manager, as there could be an MCAUser or CHLAUTH configuration which changes the user ID.

SecurityId (MQBYTE40) - input
An extension to the user ID running the application. Its length is given by MQ_SECURITY_ID_LENGTH.

In the case of a client, this is the user ID sent from the client to the server. Note, that this might not be the effective user ID the client is running against in the queue manager, as there could be an MCAUser or CHLAUTH configuration which changes the user ID.

ConnectionName (MQCHAR264) - input
The connection name field, set to the address of the client. For example, for TCP/IP, it would be the client IP address.

The length of this field is given by MQ_CONN_NAME_LENGTH.

In the case of a client, this is the partner address of the queue manager.

LongMCAUserIdLength (MQLONG) - input
The length of the long MCA user identifier.

When MCA connects to the queue manager this field is set to the length of the long MCA user identifier (or zero if there is no such identifier).

In the case of a client, this is the client long user identifier.

LongRemoteUserIdLength (MQLONG) - input
The length of the long remote user identifier.

When MCA connects to the queue manager this field is set to the length of the long remote user identifier. Otherwise this field will be set to zero

In the case of a client, set this field to zero.

LongMCAUserIdPtr (MQPTR) - input
Address of long MCA user identifier.

When MCA connects to the queue manager this field is set to the address of the long MCA user identifier (or to a null pointer if there is no such identifier).

In the case of a client, this is the client long user identifier.

LongRemoteUserIdPtr (MQPTR) - input
The address of the long remote user identifier.

When MCA connects to the queue manager this field is set to the address of the long remote user identifier (or to a null pointer if there is no such identifier).

In the case of a client, set this field to zero.

ApplName (MQCHAR28) - input
The name of the application or component that issued the WebSphere MQ API call.

The rules for generating the ApplName are the same as for generating the default name for an MQPUT.

The value of this field is found by querying the operating system for the program name. Its length is given by MQ_APPL_NAME_LENGTH.

ApplType (MQLONG) - input
The type of application or component that issued the WebSphere MQ API call.

The value is MQAT_DEFAULT for the platform on which the application is compiled, or it equates to one of the defined MQAT_* values.

The exit handler sets this field on entry to each exit function.

ProcessId (MQPID) - input
The operating system process identifier.

Where applicable, the exit handler sets this field on entry to each exit function.

ThreadId (MQTID) - input
The MQ thread identifier. This is the same identifier used in MQ trace and FFST™ dumps, but might be different from the operating system thread identifier.

Where applicable, the exit handler sets this field on entry to each exit function.

ChannelName (MQCHAR) - input
The name of the channel, padded with blanks, if applicable and known.

If not applicable, this field is set to NULL characters.

Reserved1 (MQBYTE4) - input
This field is reserved.

ChanneDefinition (PMQCD) - input
A pointer to the channel definition being used, if applicable and known.

If not applicable, this field is set to NULL characters.

Note that the pointer is only completed if the connection is processing on behalf of a WebSphere MQ channel and that channel definition has been read.

In particular, the channel definition is not given on the server when the first MQCONN call is made for the channel. Furthermore, if the pointer is filled, the structure (and any sub structures) pointed to by the pointer must be treated as read only; any updating of the structure would lead to unpredictable results and is not supported.

In the case of a client, fields other than those with a value specified for a client, contain values that are appropriate for a client application.

The exit chain area and exit chain area header (MQACH):

If required, an exit function can acquire storage for an exit chain area and set the ExitChainAreaPtr in MQAXP to point to this storage.

Exits (either the same or different exit functions) can acquire multiple exit chain areas and link them together. Exit chain areas must only be added or removed from this list while called from the exit handler. This ensures that there are no serialization issues caused by different threads adding or removing areas from the list at the same time.

An exit chain area must start with an MQACH header structure, the C declaration for which is:

```
typedef struct tagMQACH {
   MQCHAR4   StrucId;             /* Structure identifier */
   MQLONG    Version;             /* Structure version number */
   MQLONG    StrucLength;         /* Length of the MQACH structure */
   MQLONG    ChainAreaLength;     /* Exit chain area length */
   MQCHAR48  ExitInfoName         /* Exit information name */
   PMQACH    NextChainAreaPtr;    /* Pointer to next exit chain area */
 };
```

The fields in the exit chain area header are:

StrucId (MQCHAR4) - input
The exit chain area structure identifier, with an initial value, defined by MQACH_DEFAULT, of MQACH_STRUC_ID.

For C programs, the constant MQACH_STRUC_ID_ARRAY is also defined; this has the same value as MQACH_STRUC_ID, but as an array of characters instead of a string.

Version (MQLONG) - input
The structure version number, as follows:

MQACH_VERSION_1
The version number for the exit parameter structure.

MQACH_CURRENT_VERSION
The current version number for the exit context structure.

The initial value of this field, defined by MQACH_DEFAULT, is `MQACH_CURRENT_VERSION`.

Note: If you introduce a new version of this structure, the layout of the existing part does not change. Exit functions must check that the version number is equal to or greater than the lowest version containing the fields that the exit function needs to use.

StrucLength (MQLONG) - input
The length of the MQACH structure. Exits can use this field to determine the start of the exit data, setting it to the length of the structure created by the exit.

The initial value of this field, defined by MQACH_DEFAULT, is MQACH_CURRENT_LENGTH.

ChainAreaLength (MQLONG) - input
The exit chain area length, set to the overall length of the current exit chain area, including the MQACH header.

The initial value of this field, defined by MQACH_DEFAULT, is zero.

ExitInfoName (MQCHAR48) - input
The exit information name.

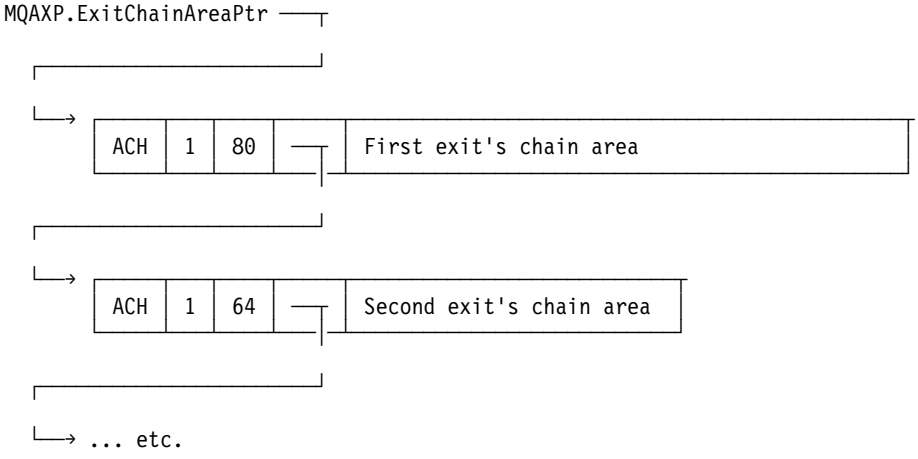
When an exit creates an MQACH structure, it must initialize this field with its own ExitInfoName, so that later this MQACH structure can be found by either another instance of this exit, or by a cooperating exit.

The initial value of this field, defined by MQACH_DEFAULT, is a zero length string ({""}).

NextChainAreaPtr (PMQACH) - input
A pointer to the next exit chain area with an initial value, defined by MQACH_DEFAULT, of null pointer (NULL).

Exit functions must release the storage for any exit chain areas that they acquire, and manipulate the chain pointers to remove their exit chain areas from the list.

An exit chain area can be constructed as follows:



External constants:

Use this topic as reference information for external constants available for API exists.

The following external constants are available for API exits:

MQXF_* (exit function identifiers)

```
MQXF_INIT                   1       X'00000001'
MQXF_TERM                   2       X'00000002'
MQXF_CONN                   3       X'00000003'
MQXF_CONNX                  4       X'00000004'
MQXF_DISC                   5       X'00000005'
MQXF_OPEN                   6       X'00000006'
MQXF_CLOSE                  7       X'00000007'
MQXF_PUT1                   8       X'00000008'
MQXF_PUT                    9       X'00000009'
MQXF_GET                    10      X'0000000A'
MQXF_DATA_CONV_ON_GET       11      X'0000000B'
MQXF_INQ                    12      X'0000000C'
MQXF_SET                    13      X'0000000D'
MQXF_BEGIN                  14      X'0000000E'
MQXF_CMIT                   15      X'0000000F'
MQXF_BACK                   16      X'00000010'
MQXF_STAT                   18      X'00000012'
MQXF_CB                     19      X'00000013'
MQXF_CTL                    20      X'00000014'
MQXF_CALLBACK               21      X'00000015'
MQXF_SUB                    22      X'00000016'
MQXF_SUBRQ                  23      X'00000017'
MQXF_XACLOSE                24      X'00000018'
MQXF_XACOMMIT               25      X'00000019'
MQXF_XACOMPLETE             26      X'0000001A'
MQXF_XAEND                  27      X'0000001B'
MQXF_XAFORGET               28      X'0000001C'
MQXF_XAOPEN                 29      X'0000001D'
MQXF_XAPREPARE              30      X'0000001E'
MQXF_XARECOVER              31      X'0000001F'
MQXF_XAROLLBACK             32      X'00000020'
MQXF_XASTART                33      X'00000021'
MQXF_AXREG                  34      X'00000022'
MQXF_AXUNREG                35      X'00000023'
```

MQXR_* (exit reasons)

```
MQXR_BEFORE                 1       X'00000001'
MQXR_AFTER                  2       X'00000002'
MQXR_CONNECTION             3       X'00000003'
```

MQXE_* (environments)

```
MQXE_OTHER                  0       X'00000000'
MQXE_MCA                    1       X'00000001'
MQXE_MCA_SVRCONN            2       X'00000002'
MQXE_COMMAND_SERVER         3       X'00000003'
MQXE_MQSC                   4       X'00000004'
```

MQ*_* (additional constants)

```
MQAXP_VERSION_1             1
MQAXP_VERSION_2             2
MQAXC_VERSION_1             1
MQACH_VERSION_1             1

MQAXP_CURRENT_VERSION       1
MQAXC_CURRENT_VERSION       1
MQACH_CURRENT_VERSION       1

MQXACT_EXTERNAL             1
MQXACT_INTERNAL             2
```

```
MQXT_API_EXIT                   2

MQACH_LENGTH_1                  68 (32-bit platforms)
                                72 (64-bit platforms)
                                80 (128-bit platforms)

MQACH_CURRENT_LENGTH            68 (32-bit platforms)
                                72 (64-bit platforms)
                                80 (128-bit platforms)
```

MQ*_* (null constants)

```
MQXPDA_NONE                     X'00...00' (48 nulls)
MQXPDA_NONE_ARRAY               '\0','\0',...,'\0','\0'
```

MQXCC_* (completion codes)

```
MQXCC_FAILED                    -8
```

MQRC_* (reason codes)

MQRC_API_EXIT_ERROR 2374 X'00000946'
An exit function invocation has returned an invalid response code, or has failed in some way, and the queue manager cannot determine the next action to take.

Examine both the ExitResponse and ExitResponse2 fields of the MQAXP to determine the bad response code, and change the exit to return a valid response code.

MQRC_API_EXIT_INIT_ERROR 2375 X'00000947'
The queue manager encountered an error while initializing the execution environment for an API exit function.

MQRC_API_EXIT_TERM_ERROR 2376 X'00000948'
The queue manager encountered an error while closing the execution environment for an API exit function.

MQRC_EXIT_REASON_ERROR 2377 X'00000949'
The value of the ExitReason field supplied on an exit entry point registration call (MQXEP) call is in error.

Examine the value of the ExitReason field to determine and correct the bad exit reason value.

MQRC_RESERVED_VALUE_ERROR 2378 X'0000094A'
The value of the Reserved field is in error.

Examine the value of the Reserved field to determine and correct the Reserved value.

C language typedefs:

This topic provides information about typedefs associated with API exits available in C language.

Here are the C language typedefs associated with the API exits:

```
typedef PMQLONG  MQPOINTER PPMQLONG;
typedef PMQBYTE  MQPOINTER PPMQBYTE;
typedef PMQHOBJ  MQPOINTER PPMQHOBJ;
typedef PMQOD    MQPOINTER PPMQOD;
typedef PMQMD    MQPOINTER PPMQMD;
typedef PMQPMO   MQPOINTER PPMQPMO;
typedef PMQGMO   MQPOINTER PPMQGMO;
typedef PMQCNO   MQPOINTER PPMQCNO;
typedef PMQBO    MQPOINTER PPMQBO;

typedef MQAXP    MQPOINTER PMQAXP;
typedef MQACH    MQPOINTER PMQACH;
typedef MQAXC    MQPOINTER PMQAXC;
```

```
typedef MQCHAR   MQCHAR16[16];
typedef MQCHAR16 MQPOINTER PMQCHAR16;

typedef MQLONG   MQPID;
typedef MQLONG   MQTID;
```

The exit entry point registration call (MQXEP):

Use this information to learn about MQXEP, MQXEP C language invocation, and MQXEP C function prototype.

Use the MQXEP call to:

1. Register the before and after WebSphere MQ API exit invocation points at which to invoke exit functions
2. Specify the exit function entry points
3. Deregister the exit function entry points

You would typically code the MQXEP calls in the MQ_INIT_EXIT exit function, but you can specify them in any subsequent exit function.

If you use an MQXEP call to register an already registered exit function, the second MQXEP call completes successfully, replacing the registered exit function.

If you use an MQXEP call to register a NULL exit function, the MQXEP call completes successfully and the exit function is deregistered.

If MQXEP calls are used to register, deregister, and reregister a particular exit function during the life of a connection request, the previously registered exit function is reactivated. Any storage still allocated and associated with this exit function instance is available for use by the exit's functions. (This storage is typically released during the invocation of the termination exit function.)

The interface to MQXEP is:

```
MQXEP (Hconfig, ExitReason, Function, EntryPoint, &ExitOpts, &CompCode, &Reason)
```

where:

Hconfig (MQHCONFIG) - input
The configuration handle, representing the API exit that includes the set of functions being initialized. This value is generated by the queue manager immediately before invoking the MQ_INIT_EXIT function, and is passed in the MQAXP to each API exit function.

ExitReason (MQLONG) - input
The reason for which the entry point is being registered, from the following reasons:

- Connection level initialization or termination (MQXR_CONNECTION)
- Before a WebSphere MQ API call (MQXR_BEFORE)
- After a WebSphere MQ API call (MQXR_AFTER)

Function (MQLONG) - input
The function identifier, valid values for which are the MQXF_* constants (see “External constants” on page 2424).

EntryPoint (PMQFUNC) - input
The address of the entry point for the exit function to be registered. The value NULL indicates either that the exit function has not been provided, or that a previous registration of the exit function is being deregistered.

ExitOpts(MQXEPO)
API exits can specify options that control how API exits are registered. If a null pointer is specified for this field, the default values of the MQXEPO structure are assumed.

CompCode (MQLONG) - output
The completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) - output
The reason code that qualifies the completion code.

If the completion code is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If the completion code is MQCC_FAILED:

MQRC_HCONFIG_ERROR
(2280, X'8E8') The supplied configuration handle is not valid. Use the configuration handle from the MQAXP.

MQRC_EXIT_REASON_ERROR
(2377, X'949') The supplied exit function invocation reason is either not valid or is not valid for the supplied exit function identifier.

Either use one of the valid exit function invocation reasons (MQXR_* value), or use a valid function identifier and exit reason combination. (See Table 240.)

MQRC_FUNCTION_ERROR
(2281, X'8E9') The supplied function identifier is not valid for API exit reason. The following table shows valid combinations of function identifiers and ExitReasons.

Table 240. Valid combinations of function identifiers and ExitReasons

Function	ExitReason
MQXF_INIT MQXF_TERM	MQXR_CONNECTION

Table 240. Valid combinations of function identifiers and ExitReasons (continued)

Function	ExitReason
MQXF_CONN MQXF_CONNX MQXF_DISC MQXF_OPEN MQXF_CLOSE MQXF_PUT1 MQXF_PUT MQXF_GET MQXF_INQ MQXF_SET MQXF_BEGIN MQXF_CMIT MQXF_BACK MQXF_STAT MQXF_CB MQXF_CTL MQXF_CALLBACK MQXF_SUB MQXF_SUBRQ	MQXR_BEFORE MQXR_AFTER
MQXF_DATA_CONV_ON_GET	MQXR_BEFORE

MQRC_RESOURCE_PROBLEM
(2102, X'836') An attempt to register or deregister an exit function has failed because of a resource problem.

MQRC_UNEXPECTED_ERROR
(2195, X'893') An attempt to register or deregister an exit function has failed unexpectedly.

MQRC_PROPERTY_NAME_ERROR
(2442, X'098A') Invalid ExitProperties name.

MQRC_XEPO_ERROR
(2507, X'09CB') Exit options structure not valid.

MQXEP C language invocation

```
MQXEP (Hconfig, ExitReason, Function, EntryPoint, &ExitOpts, &CompCode, &Reason);
```

Declaration for parameter list:

```
MQHCONFIG       Hconfig;        /* Configuration handle */
MQLONG          ExitReason;     /* Exit reason */
MQLONG          Function;       /* Function identifier */
PMQFUNC         EntryPoint;     /* Function entry point */
MQXEPO          ExitOpts;       /* Options that control the action of MQXEP */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code qualifying completion
                                   code */
```

MQXEP C function prototype

```
void MQXEP (
MQHCONFIG    Hconfig,        /* Configuration handle */
MQLONG       ExitReason,     /* Exit reason */
MQLONG       Function,       /* Function identifier */
PMQFUNC      EntryPoint,     /* Function entry point */
```

```
PMQXEPO      pExitOpts;      /* Options that control the action of MQXEP */
PMQLONG      pCompCode,      /* Address of completion code */
PMQLONG      pReason);       /* Address of reason code qualifying completion
                                code */
```

Exit functions:

This section describes how to invoke the exit functions available.

The descriptions of the individual functions start at "General rules for API exit routines." This topic begins with some general information to help you when using these function calls.

General rules for API exit routines:

Use this information to understand the general rules for API exit routines, and setting up and cleaning up the exit execution environment.

The following general rules apply when invoking API exit routines:

- In all cases, API exit functions are driven before validating API call parameters, and before any security checks (in the case of MQCONN, MQCONNX, or MQOPEN).
- The values of fields entered into and output from an exit routine are:
  - On input to a *before* WebSphere MQ API exit function, the value of a field can be set by the application program, or by a previous exit function invocation.
  - On output from a *before* WebSphere MQ API exit function, the value of a field can be left unchanged, or set to some other value by the exit function.
  - On input to an *after* WebSphere MQ API exit function, the value of a field can be the value set by the queue manager after processing the WebSphere MQ API call, or can be set to a value by a previous exit function invocation in the chain of exit functions.
  - On output from an *after* WebSphere MQ API call exit function, the value of a field can be left unchanged, or set to some other value by the exit function.
- Exit functions must communicate with the queue manager by using the ExitResponse and ExitResponse2 fields.
- The CompCode and Reason code fields communicate back to the application. The queue manager and exit functions can set the CompCode and Reason code fields.
- The MQXEP call returns new reason codes to the exit functions that call MQXEP. However, exit functions can translate these new reason codes to any existing reasons codes that existing and new applications can understand.
- Each exit function prototype has similar parameters to the API function with an extra level of indirection except for the CompCode and Reason.
- API exits can issue MQI calls (except MQDISC), but these MQI calls do not themselves invoke API exits.

Note, that whether the application is on a server or a client, you cannot predict the sequencing of the API exit calls. An API exit `BEFORE` call might not be followed immediately by an `AFTER` call.

The `BEFORE` call can be followed by another `BEFORE` call. For example:

BEFORE MQCTL
BEFORE Callback
BEFORE MQPUT
AFTER MQPUT
AFTER Callback
AFTER MQCTL

or

BEFORE XAOPEN
BEFORE MQCONNX
AFTER MQCONNX
AFTER XAOPEN

On the client, there is an exit that can modify the behavior of the MQCONN or MQCONNX call, called the `PreConnect` exit. The `PreConnect` exit can modify any of the parameters on the MQCONN or MQCONNX call including the queue manager name. The client calls this exit first and then invokes the MQCONN or MQCONNX call. Note that only the initial MQCONN or MQCONNX call invokes the API exit; any subsequent reconnect calls have no effect.

The execution environment

In general, all errors from exit functions are communicated back to the exit handler using the ExitResponse and ExitResponse2 fields in MQAXP.

These errors in turn are converted into MQCC_* and MQRC_* values and communicated back to the application in the CompCode and Reason fields. However, any errors encountered in the exit handler logic are communicated back to the application as MQCC_* and MQRC_* values in the CompCode and Reason fields.

If an MQ_TERM_EXIT function returns an error:

- The MQDISC call has already taken place
- There is no other opportunity to drive the *after* MQ_TERM_EXIT exit function (and thus perform exit execution environment cleanup)
- Exit execution environment cleanup is *not* performed

The exit cannot be unloaded as it might still be in use. Also, other registered exits further down in the exit chain for which the *before* exit was successful, will be driven in the reverse order.

Setting up the exit execution environment

While processing an explicit MQCONN or MQCONNX call, exit handling logic sets up the exit execution environment before invoking the exit initialization function (MQ_INIT_EXIT). Exit execution environment setup involves loading the exit, acquiring storage for, and initializing exit parameter structures. The exit configuration handle is also allocated.

If errors occur during this phase, the MQCONN or MQCONNX call fails with CompCode MQCC_FAILED and one of the following reason codes:

MQRC_API_EXIT_LOAD_ERROR
An attempt to load an API exit module has failed.

MQRC_API_EXIT_NOT_FOUND
An API exit function could not be found in the API exit module.

MQRC_STORAGE_NOT_AVAILABLE
An attempt to initialize the execution environment for an API exit function failed because insufficient storage was available.

MQRC_API_EXIT_INIT_ERROR
An error was encountered while initializing the execution environment for an API exit function.

Cleaning up the exit execution environment

While processing an explicit MQDISC call, or an implicit disconnect request as a result of an application ending, exit handling logic might need to clean up the exit execution environment after invoking the exit termination function (MQ_TERM_EXIT), if registered.

Cleaning up the exit execution environment involves releasing storage for exit parameter structures, possibly deleting any modules previously loaded into memory.

If errors occur during this phase, an explicit MQDISC call fails with CompCode MQCC_FAILED and the following reason code (errors are not highlighted on implicit disconnect requests):

MQRC_API_EXIT_TERM_ERROR
An error was encountered while closing the execution environment for an API exit function. The exit should *not* return any failure from the MQDISC before or after the MQ_TERM* API exit function calls.

API exits on clients:

A client uses the PreConnect exit to modify the behavior of the MQCONN and MQCONNX calls and does not support API exit properties.

PreConnect exit

On a client, the PreConnect exit can be used to look up the channel definition from a central repository, such as an LDAP server.

The PreConnect exit can also modify any parameter, or all the parameters, on an MQCONN or MQCONNX call itself, for example, the queue manager name.

In the case of client applications, the PreConnect exit must be called before the API exit because the MQCONN or MQCONNX API exit is called only once the name of the queue manager is known and this name can be changed by the PreConnect exit.

Note that only the initial MQCONN or MQCONNX call invokes the exit.

API exit properties

On a server, API exits can register an MQXEPO structure at initialization time. The MQXEPO structure contains the ExitProperties field which details the group of properties the exit is interested in. This has the effect of generating a separate message property handle which the exit can manipulate separately from any application message property handle.

On a client, API exit properties are not supported. If an attempt is made to register a property group name on a client, the function fails with a reason code of MQRC_EXIT_PROPS_NOT_SUPPORTED.

Backout - MQ_BACK_EXIT:

MQ_BACK_EXIT provides a backout exit function to perform *before* and *after* backout processing. Use function identifier MQXF_BACK with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* backout call exit functions.

The interface to this function is:

```
MQ_BACK_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_BACK_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_BACK_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Begin - MQ_BEGIN_EXIT:

MQ_BEGIN_EXIT provides a begin exit function to perform *before* and *after* MQBEGIN call processing. Use function identifier MQXF_BEGIN with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQBEGIN call exit functions.

The interface to this function is:

```
MQ_BEGIN_EXIT (&ExitParms, &ExitContext, &Hconn, &pBeginOptions, &CompCode,
               &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pBeginOptions (PMQBO)- input/output
Pointer to begin options.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
PMQBO    pBeginOptions;  /* Ptr to begin options */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_BEGIN_EXIT (&ExitParms, &ExitContext, &Hconn, &pBeginOptions, &CompCode,
               &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_BEGIN_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PPMQBO    ppBeginOptions,   /* Address of ptr to begin options */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Callback - MQ_CALLBACK_EXIT:

MQ_CALLBACK_EXIT provides an exit function to perform *before* and *after* callback processing. Use function identifier MQXF_CALLBACK with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* callback call exit functions.

The interface to this function is:

```
MQ_CALLBACK_EXIT (&ExitParms, &ExitContext, &Hconn, &pMsgDesc, &pGetMsgOpts,
 &pBuffer, &pMQCBContext)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure

ExitContext (MQAXC) - input/output
Exit context structure

Hconn (MQHCONN) - input/output
Connection handle

pMsgDesc
Message descriptor

pGetMsgOpts
Options that control the action of MQGET

pBuffer
Area to contain the message data

pMQCBContext
Context data for the callback

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
PMQMD    pMsgDesc;       /* Message descriptor */
PMQGMO   pGetMsgOpts;    /* Options that define the operation of the consumer */
PMQVOID  pBuffer;        /* Area to contain the message data */
PMQCBC   pContext;       /* Context data for the callback */
```

The queue manager then logically calls the exit as follows:

```
MQ_SUBRQ_EXIT (&ExitParms, &ExitContext, &Hconn, &pMsgDesc, &pGetMsgOpts, &pBuffer,
 &pContext);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CALLBACK_EXIT (
PMQAXP    pExitParms;    /* Exit parameter structure */
PMQAXC    pExitContext;  /* Exit context structure */
PMQHCONN  pHconn;        /* Connection handle */
```

```
PPMQMD    ppMsgDesc;     /* Message descriptor */
PPMQGMO   ppGetMsgOpts;  /* Options that define the operation of the consumer */
PPMQVOID  ppBuffer;      /* Area to conatin the message data */
PPMQCBC   ppContext;)     /* Context data for the callback */
```

Usage notes

1. The Callback exit is invoked before the consumer is invoked and after the consumer's consumer function has completed. Although the MQMD and MQGMO structures are alterable, changing the values in the before exit does not redrive the retrieval of a message from the queue as the message has already been removed from the queue to be delivered to the consumer function

Manage callback functions - MQ_CB_EXIT:

MQ_CB_EXIT provides an exit function to perform *before* and *after* the MQCB call. Use function identifier MQXF_CB with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQCB call exit functions.

The interface to this function is:

```
MQ_CB_EXIT (&ExitParms, &ExitContext, &Hconn, &Operation, &pCallbackDesc,
            &Hobj, &pMsgDesc, &pGetMsgOpts, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure

ExitContext (MQAXC) - input/output
Exit context structure

Hconn (MQHCONN) - input/output
Connection handle

Operation (MQLONG) - input/output
Operation value

pCallbackDesc (PMQCBD) - input/output
Callback descriptor

Hobj (MQHOBJ) - input/output
Object handle

pMsgDesc (PMQMD) - input/output
Message descriptor

pGetMsgOpts (PMQGMO) - input/output
Options that control the action of MQCB

CompCode (MQLONG) - input/output
Completion code

Reason (MQLONG) - input/output
Reason code qualifying CompCode

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
MQLONG   Operation;      /* Operation value. */
MQCBD    pMsgDesc;       /* Callback descriptor. */
MQHOBJ   Hobj;           /* Object handle. */
```

```
PMQMD    pMsgDesc;       /* Message descriptor */
PMQGMO   pGetMsgOpts;    /* Options that define the operation of the consumer */
PMQLONG  CompCode;       /* Completion code.
PMQLONG) Reason;         /* Reason code qualifying CompCode.
```

The queue manager then logically calls the exit as follows:

```
MQ_CB_EXIT (&ExitParms, &ExitContext, &Hconn, &Operation, &Hobj, &pMsgDesc,
           &pGetMsgOpts, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CB_EXIT (
PMQAXP     pExitParms;     /* Exit parameter structure */
PMQAXC     pExitContext;   /* Exit context structure */
PMQHCONN   pHconn;         /* Connection handle */
PMQLONG    pOperation;     /* Callback operation */
PMQHOBJ    pHobj;          /* Object handle */
PPMQMD     ppMsgDesc;      /* Message descriptor */
PPMQGMO    ppGetMsgOpts;   /* Options that control the action of MQCB */
PMQLONG    pCompCode;      /* Completion code */
PMQLONG    pReason;        /* Reason code qualifying CompCode */
```

Close - MQ_CLOSE_EXIT:

MQ_CLOSE_EXIT provides a close exit function to perform *before* and *after* MQCLOSE call processing. Use function identifier MQXF_CLOSE with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQCLOSE call exit functions.

The interface to this function is:

```
MQ_CLOSE_EXIT (&ExitParms, &ExitContext, &Hconn, &pHobj,
               &Options, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pHobj (PMQHOBJ) - input
Pointer to object handle.

Options (MQLONG)- input/output
Close options.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
PMQHOBJ         pHobj;          /* Ptr to object handle */
MQLONG          Options;        /* Close options */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_CLOSE_EXIT (&ExitParms, &ExitContext,&Hconn, &pHobj, &Options,
               &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CLOSE_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PPMQHOBJ        ppHobj,         /* Address of ptr to object handle */
PMQLONG         pOptions,       /* Address of close options */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Commit - MQ_CMIT_EXIT:

MQ_CMIT_EXIT provides a commit exit function to perform *before* and *after* commit processing. Use function identifier MQXF_CMIT with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* commit call exit functions.

If a commit operation fails, and the transaction is backed out, the MQCMIT call fails with MQCC_WARNING and MQRC_BACKED_OUT. These return and reason codes are passed into any *after* MQCMIT exit functions to give the exit functions an indication that the unit of work has been backed out.

The interface to this function is:

```
MQ_CMIT_EXIT (&ExitParms, &ExitContext, &Hconn, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_CMIT_EXIT (&ExitParms, &ExitContext,&Hconn, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CMIT_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Usage notes

1. The MQ_GET_EXIT function interface described here is used for both the MQXF_GET exit function and the "MQXF_DATA_CONV_ON_GET" on page 2445 exit function.

   Separate entry points are defined for these two exit functions, so to intercept *both* the MQXEP call must be used twice; for this call use function identifier MQXF_GET.

   Because the MQ_GET_EXIT interface is the same for MQXF_GET and MQXF_DATA_CONV_ON_GET, a single exit function can be used for both; the *Function* field in the MQAXP structure indicates which exit function has been invoked. Alternatively, the MQXEP call can be used to register different exit functions for the two cases.

Connect and connect extension - MQ_CONNX_EXIT:

MQ_CONNX_EXIT provides:

- Connection exit function to perform *before* and *after* MQCONN processing
- Connection extension exit function to perform *before* and *after* MQCONNX processing

The same interface, described here, is invoked for both MQCONN and MQCONNX call exit functions.

When the message channel agent (MCA) responds to an inbound client connection, the MCA can connect and make a number of WebSphere MQ API calls before the client state is fully known. These API calls call the API exit functions with the MQAXC based on the MCA program itself (for example in the UserId and ConnectionName fields of the MQAXC).

When the MCA responds to subsequent inbound client API calls, the MQAXC structure is based on the inbound client, setting the UserId and ConnectionName fields appropriately.

The queue manager name set by the application on an MQCONN or MQCONNX call is passed to the underlying connect call. Any attempt by a *before* MQ_CONNX_EXIT to change the name of the queue manager has no effect.

Use function identifiers MQXF_CONN and MQXF_CONNX with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQCONN and MQCONNX call exit functions.

An MQ_CONNX_EXIT exit called for reason MQXR_BEFORE *must not* issue any WebSphere MQ API calls, as the correct environment has not been set up at this time.

An MQ_CONNX_EXIT cannot call MQDISC from an API exit call for the connection for which it is being called. This restriction is applicable to both client and server API exits.

The interface to MQCONN and MQCONNX is identical:

```
MQ_CONNX_EXIT (&ExitParms, &ExitContext, &pQMgrName, &pConnectOpts,
               &pHconn, &CompCode, &Reason);
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

pQMgrName (PMQCHAR) - input
Pointer to the queue manager name supplied on the MQCONNX call. The exit must not change this name on the MQCONN or MQCONNX call.

pConnectOpts (PMQCNO) - input/output
Pointer to the options that control the action of the MQCONNX call.

See "MQCNO - Connect options" on page 1607 for details.

For exit function MQXF_CONN, pConnectOpts points to the default connect options structure (MQCNO_DEFAULT).

pHconn (PMQHCONN) - input
Pointer to the connection handle.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion)

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
PMQCHAR         pQMgrName;      /* Ptr to Queue manager name */
PMQCNO          pConnectOpts;   /* Ptr to Connection options */
PMQHCONN        pHconn;         /* Ptr to Connection handle */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_CONNX_EXIT (&ExitParms, &ExitContext, &pQMgrName, &pConnectOps,
               &pHconn, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CONNX_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PPMQCHAR        ppQMgrName,     /* Address of ptr to queue manager name */
PPMQCNO         ppConnectOpts,  /* Address of ptr to connection options */
PPMQHCONN       ppHconn,        /* Address of ptr to connection handle */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

1. The MQ_CONNX_EXIT function interface described here is used for both the MQCONN call and the MQCONNX call. However, separate entry points are defined for these two calls. To intercept *both* calls, the MQXEP call must be used at least twice - once with function identifier MQXF_CONN, and again with MQXF_CONNX.

   Because the MQ_CONNX_EXIT interface is the same for MQCONN and MQCONNX, a single exit function can be used for both calls; the *Function* field in the MQAXP structure indicates which call is in progress. Alternatively, the MQXEP call can be used to register different exit functions for the two calls.
2. When a message channel agent (MCA) responds to an inbound client connection, the MCA can issue a number of MQ calls before the client state is fully known. These MQ calls result in the API exit functions being invoked with the MQAXC structure containing data relating to the MCA, and not to

the client (for example, user identifier and connection name). However, once the client state is fully known, subsequent MQ calls result in the API exit functions being invoked with the appropriate client data in the MQAXC structure.

3. All MQXR_BEFORE exit functions are invoked before any parameter validation is performed by the queue manager. The parameters might therefore be invalid (including invalid pointers for the addresses of parameters).

   The MQ_CONNX_EXIT function is invoked before any authorization checks are performed by the queue manager.

4. The exit function must not change the name of the queue manager specified on the MQCONN or MQCONNX call. If the name is changed by the exit function, the results are undefined.
5. An MQXR_BEFORE exit function for the MQ_CONNX_EXIT cannot issue MQ calls other than MQXEP.

Control callback - MQ_CTL_EXIT:

MQ_CTL_EXIT provides a subscription request exit function to perform *before* and *after* control callback processing. Use function identifier MQXF_CTL with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* control callback call exit functions.

The interface to this function is:

```
MQ_CTL_EXIT (&Hconn, &Operation, &ControlOpts, &CompCode, &Reason)
```

where the parameters are:

Hconn (MQHCONN) - input/output
Connection handle.

Operation (MQLONG) input/output
The operation being processed on the callback defined for the specified object handle

ControlOpts (MQCTLO) input/output
Options that control the action of MQCTL

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQHCONN  Hconn;          /* Connection handle */
MQLONG   Operation;      /* Operation being processed */
MQCTLO   ControlOpts;    /* Options that control the action of MQCTL  */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_CTL_EXIT (&Hconn, &Operation, &ControlOpts, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_CTL_EXIT (
PMQHCONN  pHconn;          /* Address of connection handle */
PMQLONG   pOperation;      /* Address of operation being processed */
PMQCTLO   pControlOpts;    /* Address of options that control the action of MQCTL  */
PMQLONG   pCompCode;       /* Address of completion code */
PMQLONG   pReason;)         /* Address of reason code qualifying completion code */
```

Disconnect - MQ_DISC_EXIT:

MQ_DISC_EXIT provides a disconnect exit function to perform *before* and *after* MQDISC exit processing. Use function identifier MQXF_DISC with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQDISC call exit functions.

The interface to this function is

```
MQ_DISC_EXIT (&ExitParms, &ExitContext, &pHconn,
              &CompCode, &Reason);
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

pHconn (PMQHCONN) - input
Pointer to the connection handle.

For the before MQDISC call, the value of this field is one of:

- The connection handle returned on the MQCONN or MQCONNX call
- Zero, for environments where an environment-specific adapter has connected to the queue manager
- A value set by a previous exit function invocation

For the after MQDISC call, the value of this field is zero or a value set by a previous exit function invocation.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
PMQHCONN        pHconn;         /* Ptr to Connection handle */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_DISC_EXIT (&ExitParms, &ExitContext, &pHconn,
              &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_DISC_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PPMQHCONN       ppHconn,        /* Address of ptr to connection handle */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Get - MQ_GET_EXIT:

MQ_GET_EXIT provides a get exit function to perform *before* and *after* MQGET call processing.

There are two function identifiers:

1. Use MQXF_GET with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQGET call exit functions.
2. See "MQXF_DATA_CONV_ON_GET" on page 2445 for information on using the MQXF_DATA_CONV_ON_GET function identifier.

The interface to this function is:

```
MQ_GET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc,
             &pGetMsgOpts, &BufferLength, &pBuffer, &pDataLength,
             &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Hobj (MQHOBJ) - input/output
Object handle.

pMsgDesc (PMQMD) - input/output
Pointer to message descriptor.

pGetMsgOpts (PMQGMO) - input/output
Pointer to get message options.

BufferLength (MQLONG) - input/output
Message buffer length.

pBuffer (PMQBYTE) - input/output
Pointer to message buffer.

pDataLength (PMQLONG) - input/output
Pointer to data length field.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
MQHOBJ          Hobj;           /* Object handle */
PMQMD           pMsgDesc;       /* Ptr to message descriptor */
PMQPMO          pGetMsgOpts;    /* Ptr to get message options */
MQLONG          BufferLength;   /* Message buffer length */
PMQBYTE         pBuffer;        /* Ptr to message buffer */
PMQLONG         pDataLength;    /* Ptr to data length field */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_GET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc,
             &pGetMsgOpts, &BufferLength, &pBuffer, &pDataLength,
             &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_GET_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PMQHOBJ         pHobj,          /* Address of object handle */
```

```
PPMQMD          ppMsgDesc,      /* Address of ptr to message descriptor */
PPMQGMO         ppGetMsgOpts,   /* Address of ptr to get message options */
PMQLONG         pBufferLength,  /* Address of message buffer length */
PPMQBYTE        ppBuffer,       /* Address of ptr to message buffer */
PPMQLONG        ppDataLength,   /* Address of ptr to data length field */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

1. The MQ_GET_EXIT function interface described here is used for both the MQXF_GET exit function and the "MQXF_DATA_CONV_ON_GET" exit function.

   Separate entry points are defined for these two exit functions, so to intercept *both* the MQXEP call must be used twice; for this call use function identifier MQXF_GET.

   Because the MQ_GET_EXIT interface is the same for MQXF_GET and MQXF_DATA_CONV_ON_GET, a single exit function can be used for both; the *Function* field in the MQAXP structure indicates which exit function has been invoked. Alternatively, the MQXEP call can be used to register different exit functions for the two cases.

MQXF_DATA_CONV_ON_GET:
See MQ_GET_EXIT for information about the interface to this call, and a sample C language declaration.

Usage notes

If registered, this entry point is called when messages arrive at the application but before any data conversion has occurred. This can be useful if the API exit needs to perform processing, such as decryption or decompression, before the message is passed to data conversion. The exit can, if necessary, cause data conversion to be bypassed by returning MQXCC_SUPPRESS_FUNCTION; for more information, see MQAXP structure.

Registering for this entry point on a client has the effect of causing the data conversion to be performed locally on the client machine. For correct operation it might, therefore, be necessary to install the application conversion exits on the client. Note that MQXF_DATA_CONV_ON_GET is also used for asynchronous consume.

When using the MQ_GET_EXIT call, use MQXF_DATA_CONV_ON_GET, with exit reason MQXR_BEFORE, to register a *before* MQGET data conversion exit function.

There is no MQXR_AFTER exit function for MQXF_DATA_CONV_ON_GET; the MQXR_AFTER exit function for MQXF_GET provides the required capability for exit processing after data conversion.

Separate entry points are defined for the MQ_GET_EXIT call, so to intercept *both* exit functions, the MQXEP call must be used twice; for this call use function identifier MQXF_DATA_CONV_ON_GET.

Because the MQ_GET_EXIT interface is the same for MQXF_GET and MQXF_DATA_CONV_ON_GET, a single exit function can be used for both; the *Function* field in the MQAXP structure indicates which exit function has been invoked. Alternatively, the MQXEP call can be used to register different exit functions for the two cases.

Initialization - MQ_INIT_EXIT:

MQ_INIT_EXIT provides connection level initialization, indicated by setting ExitReason in MQAXP to MQXR_CONNECTION.

During the initialization, note the following:

- The MQ_INIT_EXIT function calls MQXEP to register the WebSphere MQ API verbs and the ENTRY and EXIT points in which it is interested.
- Exits do not need to intercept all the WebSphere MQ API verbs. Exit functions are invoked only if an interest has been registered.
- Storage that is to be used by the exit can be acquired while initializing it.
- If a call to this function fails, the MQCONN or MQCONNX call that invoked it also fails with a CompCode and Reason that depend on the value of the ExitResponse field in MQAXP.
- An MQ_INIT_EXIT exit must not issue WebSphere MQ API calls, because the correct environment has not been set up at this time.
- If an MQ_INIT_EXIT fails with MQXCC_FAILED, the queue manager returns from the MQCONN or MQCONNX call that called it with MQCC_FAILED and MQRC_API_EXIT_ERROR.
- If the queue manager encounters an error while initializing the API exit function execution environment before invoking the first MQ_INIT_EXIT, the queue manager returns from the MQCONN or MQCONNX call that invoked MQ_INIT_EXIT with MQCC_FAILED and MQRC_API_EXIT_INIT_ERROR.

The interface to MQ_INIT_EXIT is:

```
MQ_INIT_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

CompCode (MQLONG) - input/output
Pointer to completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Pointer to reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

The CompCode and Reason returned to the application depend on the value of the ExitResponse field in MQAXP.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_INIT_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_INIT_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

1. The MQ_INIT_EXIT function can issue the MQXEP call to register the addresses of the exit functions for the particular MQ calls to be intercepted. It is not necessary to intercept all MQ calls, or to intercept both MQXR_BEFORE and MQXR_AFTER calls. For example, an exit suite could choose to intercept only the MQXR_BEFORE call of MQPUT.
2. Storage that is to be used by exit functions in the exit suite can be acquired by the MQ_INIT_EXIT function. Alternatively, exit functions can acquire storage when they are invoked, as and when needed. However, all storage should be freed before the exit suite is terminated; the MQ_TERM_EXIT function can free the storage, or an exit function invoked earlier.
3. If MQ_INIT_EXIT returns MQXCC_FAILED in the *ExitResponse* field of MQAXP, or fails in some other way, the MQCONN or MQCONNX call that caused MQ_INIT_EXIT to be invoked also fails, with the *CompCode* and *Reason* parameters set to appropriate values.
4. An MQ_INIT_EXIT function cannot issue MQ calls other than MQXEP.

Inquire - MQ_INQ_EXIT:

MQ_INQ_EXIT provides an inquire exit function to perform *before* and *after* MQINQ call processing. Use function identifier MQXF_INQ with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQINQ call exit functions.

The interface to this function is:

```
MQ_INQ_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount,
             &pSelectors, &IntAttrCount, &pIntAttrs, &CharAttrLength,
             &pCharAttrs, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Hobj (MQHOBJ) - input
Object handle.

SelectorCount (MQLONG) - input
Count of selectors

pSelectors (PMQLONG) - input/output
Pointer to array of selector values.

IntAttrCount (MQLONG) - input
Count of integer attributes.

pIntAttrs (PMQLONG) - input/output
Pointer to array of integer attribute values.

CharAttrLength (MQLONG) - input/output
Character attributes array length.

pCharAttrs (PMQCHAR) - input/output
Pointer to character attributes array.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;       /* Exit parameter structure */
MQAXC    ExitContext;     /* Exit context structure */
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
PMQLONG  pSelectors;      /* Ptr to array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
PMQLONG  pIntAttrs;       /* Ptr to array of integer attributes */
MQLONG   CharAttrLength;  /* Length of char attributes array */
PMQCHAR  pCharAttrs;      /* Ptr to character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_INQ_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount,
             &pSelectors, &IntAttrCount, &pIntAttrs, &CharAttrLength,
             &pCharAttrs, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_INQ_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQHOBJ   pHobj,            /* Address of object handle */
PMQLONG   pSelectorCount,   /* Address of selector count */
PPMQLONG  ppSelectors,      /* Address of ptr to array of selectors */
PMQLONG   pIntAttrCount;    /* Address of count of integer attributes */
PPMQLONG  ppIntAttrs,       /* Address of ptr to array of integer attributes */
PMQLONG   pCharAttrLength,  /* Address of character attribute length */
PPMQCHAR  ppCharAttrs,      /* Address of ptr to character attributes array */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Open - MQ_OPEN_EXIT:

MQ_OPEN_EXIT provides an open exit function to perform *before* and *after* MQOPEN call processing. Use function identifier MQXF_OPEN with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQOPEN call exit functions.

The interface to this function is

```
MQ_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &Options,
              &pHobj, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pObjDesc (PMQOD) - input/output
Pointer to object descriptor.

Options (MQLONG) - input/output
Open options.

pHobj (PMQHOBJ) - input
Pointer to object handle.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
PMQOD           pObjDesc;       /* Ptr to object descriptor */
MQLONG          Options;        /* Open options */
PMQHOBJ         pHobj;          /* Ptr to object handle */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &Options,
              &pHobj, &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_OPEN_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PPMQOD          ppObjDesc,      /* Address of ptr to object descriptor */
PMQLONG         pOptions,       /* Address of open options */
PPMQHOBJ        ppHobj,         /* Address of ptr to object handle */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Put - MQ_PUT_EXIT:

MQ_PUT_EXIT provides a put exit function to perform *before* and *after* MQPUT call processing. Use function identifier MQXF_PUT with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQPUT call exit functions.

The interface to this function is:

```
MQ_PUT_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc,
             &pPutMsgOpts, &BufferLength, &pBuffer, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Hobj (MQHOBJ) - input/output
Object handle.

pMsgDesc (PMQMD) - input/output
Pointer to message descriptor.

pPutMsgOpts (PMQPMO) - input/output
Pointer to put message options.

BufferLength (MQLONG) - input/output
Message buffer length.

pBuffer (PMQBYTE) - input/output
Pointer to message buffer.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
MQHOBJ          Hobj;           /* Object handle */
PMQMD           pMsgDesc;       /* Ptr to message descriptor */
PMQPMO          pPutMsgOpts;    /* Ptr to put message options */
MQLONG          BufferLength;   /* Message buffer length */
PMQBYTE         pBuffer;        /* Ptr to message data */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_PUT_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &pMsgDesc,
             &pPutMsgOpts, &BufferLength, &pBuffer, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_PUT_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PMQHOBJ         pHobj,          /* Address of object handle */
PPMQMD          ppMsgDesc,      /* Address of ptr to message descriptor */
PPMQPMO         ppPutMsgOpts,   /* Address of ptr to put message options */
PMQLONG         pBufferLength,  /* Address of message buffer length */
PPMQBYTE        ppBuffer,       /* Address of ptr to message buffer */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

- Report messages generated by the queue manager skip the normal call processing. As a result, such messages cannot be intercepted by the MQ_PUT_EXIT function or the MQPUT1 function. However, report messages generated by the message channel agent are processed normally, and hence can be intercepted by the MQ_PUT_EXIT function or the MQ_PUT1_EXIT function. To be sure to intercepting all of the report messages generated by the MCA, both MQ_PUT_EXIT and MQ_PUT1_EXIT should be used.

Put1 - MQ_PUT1_EXIT:

MQ_PUT1_EXIT provides a *put one message only* exit function to perform *before* and *after* MQPUT1 call processing. Use function identifier MQXF_PUT1 with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQPUT1 call exit functions.

The interface to this function is:

```
MQ_PUT1_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &pMsgDesc,
              &pPutMsgOpts, &BufferLength, &pBuffer, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pObjDesc (PMQOD) - input/output
Pointer to object descriptor.

pMsgDesc (PMQMD) - input/output
Pointer to message descriptor.

pPutMsgOpts (PMQPMO) - input/output
Pointer to put message options.

BufferLength (MQLONG) - input/output
Message buffer length.

pBuffer (PMQBYTE) - input/output
Pointer to message buffer.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQHCONN         Hconn;          /* Connection handle */
PMQOD           pObjDesc;       /* Ptr to object descriptor */
PMQMD           pMsgDesc;       /* Ptr to message descriptor */
PMQPMO          pPutMsgOpts;    /* Ptr to put message options */
MQLONG          BufferLength;   /* Message buffer length */
PMQBYTE         pBuffer;        /* Ptr to message data */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_PUT1_EXIT (&ExitParms, &ExitContext, &Hconn, &pObjDesc, &pMsgDesc,
              &pPutMsgOpts, &BufferLength, &pBuffer, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_PUT1_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQHCONN        pHconn,         /* Address of connection handle */
PPMQOD          ppObjDesc,      /* Address of ptr to object descriptor */
PPMQMD          ppMsgDesc,      /* Address of ptr to message descriptor */
PPMQPMO         ppPutMsgOpts,   /* Address of ptr to put message options */
PMQLONG         pBufferLength,  /* Address of message buffer length */
PPMQBYTE        ppBuffer,       /* Address of ptr to message buffer */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Set - MQ_SET_EXIT:

MQ_SET_EXIT provides a set exit function to perform *before* and *after* MQSET call processing. Use function identifier MQXF_SET with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQSET call exit functions.

The interface to this function is:

```
MQ_SET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount,
             &pSelectors, &IntAttrCount, &pIntAttrs, &CharAttrLength,
             &pCharAttr, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Hobj (MQHOBJ) - input
Object handle.

SelectorCount (MQLONG) - input
Count of selectors

pSelectors (PMQLONG) - input/output
Pointer to array of selector values.

IntAttrCount (MQLONG) - input
Count of integer attributes.

pIntAttrs (PMQLONG) - input/output
Pointer to array of integer attribute values.

CharAttrLength (MQLONG) - input/output
Character attributes array length.

pCharAttrs (PMQCHAR) - input/output
Pointer to character attribute values.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;       /* Exit parameter structure */
MQAXC    ExitContext;     /* Exit context structure */
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
PMQLONG  pSelectors;      /* Ptr to array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
PMQLONG  pIntAttrs;       /* Ptr to array of integer attributes */
MQLONG   CharAttrLength;  /* Length of char attributes array */
PMQCHAR  pCharAttrs;      /* Ptr to character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_SET_EXIT (&ExitParms, &ExitContext, &Hconn, &Hobj, &SelectorCount,
             &pSelectors, &IntAttrCount, &pIntAttrs, &CharAttrLength,
             &pCharAttrs, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_SET_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
```

```
PMQHOBJ   pHobj,           /* Address of object handle */
PMQLONG   pSelectorCount,  /* Address of selector count */
PPMQLONG  ppSelectors,     /* Address of ptr to array of selectors */
PMQLONG   pIntAttrCount;   /* Address of count of integer attributes */
PPMQLONG  ppIntAttrs,      /* Address of ptr to array of integer attributes */
PMQLONG   pCharAttrLength, /* Address of character attribute length */
PPMQCHAR  ppCharAttrs,     /* Address of ptr to character attributes array */
PMQLONG   pCompCode,       /* Address of completion code */
PMQLONG   pReason);        /* Address of reason code qualifying completion
                              code */
```

Status - MQ_STAT_EXIT:

MQ_STAT_EXIT provides a status exit function to perform *before* and *after* MQSTAT call processing. Use function identifier MQXF_STAT with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* MQSTAT call exit functions.

The interface to this function is:

```
MQ_STAT_EXIT (&ExitParms, &ExitContext, &Hconn, &Type, &pStatus
              &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

Type (MQLONG) - input
Type of status information to retrieve.

pStatus (PMQSTS) - output
Pointer to status buffer.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

Your exit must match the following C function prototype:

```
void MQENTRY MQ_STAT_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PMQLONG   pType             /* Address of status type */
PPMQSTS   ppStatus          /* Address of status buffer */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

Termination - MQ_TERM_EXIT:

MQ_TERM_EXIT provides connection level termination, registered with a function identifier of MQXF_TERM and ExitReason MQXR_CONNECTION. If registered, MQ_TERM_EXIT is called once for every disconnect request.

As part of the termination, storage no longer required by the exit can be released, and any clean up required can be performed.

If an MQ_TERM_EXIT fails with MQXCC_FAILED, the queue manager returns from the MQDISC that called it with MQCC_FAILED and MQRC_API_EXIT_ERROR.

If the queue manager encounters an error while terminating the API exit function execution environment after invoking the last MQ_TERM_EXIT, the queue manager returns from the MQDISC call that invoked MQ_TERM_EXIT with MQCC_FAILED and MQRC_API_EXIT_TERM_ERROR

The interface to this function is:

```
MQ_TERM_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED, the exit function can set the reason code field to any valid MQRC_* value.

The CompCode and Reason returned to the application depend on the value of the ExitResponse field in MQAXP.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP           ExitParms;      /* Exit parameter structure */
MQAXC           ExitContext;    /* Exit context structure */
MQLONG          CompCode;       /* Completion code */
MQLONG          Reason;         /* Reason code */
```

The queue manager then logically calls the exit as follows:

```
MQ_TERM_EXIT (&ExitParms, &ExitContext, &CompCode, &Reason)
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_TERM_EXIT (
PMQAXP          pExitParms,     /* Address of exit parameter structure */
PMQAXC          pExitContext,   /* Address of exit context structure */
PMQLONG         pCompCode,      /* Address of completion code */
PMQLONG         pReason);       /* Address of reason code qualifying
                                   completion code */
```

Usage notes

1. The MQ_TERM_EXIT function is optional. It is not necessary for an exit suite to register a termination exit if there is no termination processing to be done.

   If functions belonging to the exit suite acquire resources during the connection, an MQ_TERM_EXIT function is a convenient point at which to free those resources, for example, freeing storage obtained dynamically.
2. If an MQ_TERM_EXIT function is registered when the MQDISC call is issued, the exit function is invoked after all of the MQDISC exit functions have been invoked.
3. If MQ_TERM_EXIT returns MQXCC_FAILED in the *ExitResponse* field of MQAXP, or fails in some other way, the MQDISC call that caused MQ_TERM_EXIT to be invoked also fails, with the *CompCode* and *Reason* parameters set to appropriate values.

Register subscription - MQ_SUB_EXIT:

MQ_SUB_EXIT provides an exit function to perform *before* and *after* subscription reregistration processing. Use function identifier MQXF_SUB with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* subscription registrationcall exit functions.

The interface to this function is:

```
MQ_SUB_EXIT (&ExitParms, &ExitContext, &Hconn, &pSubDesc, &pHobj, &pHsub, &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input/output
Connection handle.

pSubDesc - input/output
Array of attribute selectors.

pHobj - input/output
Object handle

pHsub (MQHOBJ) input/output
Subscription handle

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
PMQSD    pSubDesc;       /* Subscription descriptor */
PMQHOBJ  pHobj;          /* Object Handle */
PMQHOBJ  pHsub;          /* Subscription handle */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_SUB_EXIT (&ExitParms, &ExitContext, &Hconn, &pSubDesc, &pHobj, &pHsub,
 &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
PMQAXP    pExitParms;      /* Exit parameter structure */
PMQAXC    pExitContext;    /* Exit context structure */
PMQHCONN  pHconn;          /* Connection handle */
PPMQSD    ppSubDesc;       /* Subscription descriptor */
PPMQHOBJ  ppHobj;          /* Object Handle */
PPMQHOBJ  ppHsub;          /* Subscription handle */
PMQLONG   pCompCode;       /* Completion code */
PMQLONG   pReason;         /* Reason code qualifying completion code */
```

Subscription request - MQ_SUBRQ_EXIT:

MQ_SUBRQ_EXIT provides a subscription request exit function to perform *before* and *after* subscription request processing. Use function identifier MQXF_SUBRQ with exit reasons MQXR_BEFORE and MQXR_AFTER to register *before* and *after* subscription request call exit functions.

The interface to this function is:

```
MQ_SUBRQ_EXIT (&ExitParms, &ExitContext, &Hconn, &pHsub, &Action, &pSubRqOpts,
 &CompCode, &Reason)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input/output
Connection handle.

pHsub (MQHOBJ) input/output
Subscription handle

Action (MQLONG) input/output
Action

pSubRqOpts (MQSRO) input/output

CompCode (MQLONG) - input/output
Completion code, valid values for which are:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed

Reason (MQLONG) - input/output
Reason code qualifying the completion code.

If the completion code is MQCC_OK, the only valid value is:

MQRC_NONE
(0, x'000') No reason to report.

If the completion code is MQCC_FAILED or MQCC_WARNING, the exit function can set the reason code field to any valid MQRC_* value.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;      /* Exit parameter structure */
MQAXC    ExitContext;    /* Exit context structure */
MQHCONN  Hconn;          /* Connection handle */
PMQLONG  pHsub;          /* Subscription handle */
MQLONG   Action;         /* Action */
PMQSRO   pSubRqOpts;     /* Subscription Request Options */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying completion code */
```

The queue manager then logically calls the exit as follows:

```
MQ_SUBRQ_EXIT (&ExitParms, &ExitContext, &Hconn, &pHsub, &Action, &pSubRqOpts,
 &CompCode, &Reason);
```

Your exit must match the following C function prototype:

```
void MQENTRY MQ_SUBRQ_EXIT (
PMQAXP    pExitParms,       /* Address of exit parameter structure */
PMQAXC    pExitContext,     /* Address of exit context structure */
PMQHCONN  pHconn,           /* Address of connection handle */
PPMQHOBJ  ppHsub;           /* Address of Subscription handle */
PMQLONG   pAction;          /* Address of Action */
```

```
PPMQSRO   ppSubRqOpts;      /* Address of Subscription Request Options */
PMQLONG   pCompCode,        /* Address of completion code */
PMQLONG   pReason);         /* Address of reason code qualifying completion
                               code */
```

xa_close - XA_CLOSE_EXIT:

XA_CLOSE_EXIT provides an xa_close exit function to perform before and after xa_close processing. Use function identifier MQXF_XACLOSE with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_close call exit functions.

The interface to this function is:

```
XA_CLOSE_EXIT (&ExitParms, &ExitContext, &Hconn, &pXa_info, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXa_info (PMQCHAR) - input/output
Instance-specific resource manager information.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;    /* Exit parameter structure */
MQAXC    ExitContext;  /* Exit context structure */
MQHCONN  Hconn;        /* Connection handle */
PMQCHAR  pXa_info;     /* Instance-specific RM info */
MQLONG   Rmid;         /* Resource manager identifier */
MQLONG   Flags;        /* Resource manager options*/
MQLONG   XARetCode;    /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_CLOSE_EXIT (&ExitParms, &ExitContext, &Hconn, &pXa_info, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_CLOSE_EXIT (
     PMQAXP    pExitParms,    /* Address of exit parameter structure */
     PMQAXC    pExitContext,  /* Address of exit context structure */
     PMQHCONN  pHconn,        /* Address of connection handle */
     PPMQCHAR  ppXa_info,     /* Address of instance-specific RM info */
     PMQLONG   pRmid,         /* Address of resource manager identifier */
     PMQLONG   pFlags,        /* Address of resource manager options*/
     PMQLONG   pXARetCode);   /* Address of response from XA call */
```

xa_commit - XA_COMMIT_EXIT:

XA_COMMIT_EXIT provides an xa_commit exit function to perform before and after xa_commit processing. Use function identifier MQXF_XACOMMIT with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_commit call exit functions.

The interface to this function is:

```
XA_COMMIT_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP    ExitParms;   /* Exit parameter structure */
MQAXC    ExitContext; /* Exit context structure */
MQHCONN  Hconn;       /* Connection handle */
MQPTR    pXID;        /* Transaction branch ID */
MQLONG   Rmid;        /* Resource manager identifier */
MQLONG   Flags;       /* Resource manager options*/
MQLONG   XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_COMMIT_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_COMMIT_EXIT (
     PMQAXP    pExitParms,   /* Address of exit parameter structure */
     PMQAXC    pExitContext, /* Address of exit context structure */
     PMQHCONN  pHconn,       /* Address of connection handle */
     PMQPTR    ppXID,        /* Address of transaction branch ID */
     PMQLONG   pRmid,        /* Address of resource manager identifier */
     PMQLONG   pFlags,       /* Address of resource manager options*/
     PMQLONG   pXARetCode);  /* Address of response from XA call */
```

xa_complete - XA_COMPLETE_EXIT:

XA_COMPLETE_EXIT provides an xa_complete exit function to perform before and after xa_complete processing. Use function identifier MQXF_XACOMPLETE with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_complete call exit functions.

The interface to this function is:

```
XA_COMPLETE_EXIT (&ExitParms, &ExitContext, &Hconn, &pHandle, &pRetval, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pHandle (PMQLONG) - input/output
Pointer to asynchronous operation.

pRetVal (PMQLONG) - input/output
Return value of asynchronous operation.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
PMQLONG pHandle;     /* Ptr to asynchronous op */
PMQLONG pRetval;     /* Return value of async op */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_COMPLETE_EXIT (&ExitParms, &ExitContext, &Hconn, &pHandle, &pRetval, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_COMPLETE_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PPMQLONG ppHandle,     /* Address of ptr to asynchronous op */
     PPMQLONG ppRetval,     /* Address of return value of async op */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_end - XA_END_EXIT:

XA_END_EXIT provides an xa_end exit function to perform before and after xa_end processing. Use function identifier MQXF_XAEND with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_end call exit functions.

The interface to this function is:

```
XA_END_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_END_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_END_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_forget - XA_FORGET_EXIT:

XA_FORGET_EXIT provides an xa_forget exit function to perform before and after xa_forget processing. Use function identifier MQXF_XAFORGET with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_forget call exit functions.

The interface to this function is:

```
XA_FORGET_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_FORGET_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_FORGET_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_open - XA_OPEN_EXIT:

XA_OPEN_EXIT provides an xa_open exit function to perform before and after xa_open processing. Use function identifier MQXF_XAOPEN with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_open call exit functions.

The interface to this function is:

```
XA_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pXa_info, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXa_info (PMQCHAR) - input/output
Instance-specific resource manager information.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
PMQCHAR pXa_info;    /* Instance-specific RM info */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_OPEN_EXIT (&ExitParms, &ExitContext, &Hconn, &pXa_info, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_OPEN_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PPMQCHAR ppXa_info,    /* Address of instance-specific RM info */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_prepare - XA_PREPARE_EXIT:

XA_PREPARE_EXIT provides an xa_prepare exit function to perform before and after xa_prepare processing. Use function identifier MQXF_XAPREPARE with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_prepare call exit functions.

The interface to this function is:

```
XA_PREPARE_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_PREPARE_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_PREPARE_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_recover - XA_RECOVER_EXIT:

XA_RECOVER_EXIT provides an xa_recover exit function to perform before and after xa_recover processing. Use function identifier MQXF_XARECOVER with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_recover call exit functions.

The interface to this function is:

```
XA_RECOVER_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Count, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Count (MQLONG) - input/output
Maximum XIDs in XID array

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Count;       /* Max XIDs in XID array */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_RECOVER_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Count, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_RECOVER_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pCount,       /* Address of max XIDs in XID array */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_rollback - XA_ROLLBACK_EXIT:

XA_ROLLBACK_EXIT provides an xa_rollback exit function to perform before and after xa_rollback processing. Use function identifier MQXF_XAROLLBACK with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_rollback call exit functions.

The interface to this function is:

```
XA_ROLLBACK_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_ROLLBACK_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_ROLLBACK_EXIT (
     PMQAXP   pExitParms,   /* Address of exit parameter structure */
     PMQAXC   pExitContext, /* Address of exit context structure */
     PMQHCONN pHconn,       /* Address of connection handle */
     PMQPTR   ppXID,        /* Address of transaction branch ID */
     PMQLONG  pRmid,        /* Address of resource manager identifier */
     PMQLONG  pFlags,       /* Address of resource manager options*/
     PMQLONG  pXARetCode);  /* Address of response from XA call */
```

xa_start - XA_START_EXIT:

XA_START_EXIT provides an xa_start exit function to perform before and after xa_start processing. Use function identifier MQXF_XASTART with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after xa_start call exit functions.

The interface to this function is:

```
XA_START_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP   ExitParms;   /* Exit parameter structure */
MQAXC   ExitContext; /* Exit context structure */
MQHCONN Hconn;       /* Connection handle */
MQPTR   pXID;        /* Transaction branch ID */
MQLONG  Rmid;        /* Resource manager identifier */
MQLONG  Flags;       /* Resource manager options*/
MQLONG  XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
XA_START_EXIT (&ExitParms, &ExitContext, &Hconn, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY XA_START_EXIT (
MQAXP   pExitParms,   /* Address of exit parameter structure */
MQAXC   pExitContext, /* Address of exit context structure */
MQHCONN pHconn,       /* Address of connection handle */
MQPTR   ppXID,        /* Address of transaction branch ID */
MQLONG  pRmid,        /* Address of resource manager identifier */
MQLONG  pFlags,       /* Address of resource manager options*/
MQLONG  pXARetCode);  /* Address of response from XA call */
```

ax_reg - AX_REG_EXIT:

AX_REG_EXIT provides an ax_reg exit function to perform before and after ax_reg processing. Use function identifier MQXF_AXREG with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after ax_reg call exit functions.

The interface to this function is:

```
AX_REG_EXIT (&ExitParms, &ExitContext, &pXID, &Rmid, &Flags, &XARetCode)
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Hconn (MQHCONN) - input
Connection handle.

pXID (MQPTR) - input/output
Transaction branch ID.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP  ExitParms;   /* Exit parameter structure */
MQAXC  ExitContext; /* Exit context structure */
MQPTR  pXID;        /* Transaction branch ID */
MQLONG Rmid;        /* Resource manager identifier */
MQLONG Flags;       /* Resource manager options*/
MQLONG XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
AX_REG_EXIT (&ExitParms, &ExitContext, &pXID, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY AX_REG_EXIT (
     PMQAXP  pExitParms,   /* Address of exit parameter structure */
     PMQAXC  pExitContext, /* Address of exit context structure */
     PMQPTR  ppXID,        /* Address of transaction branch ID */
     PMQLONG pRmid,        /* Address of resource manager identifier */
     PMQLONG pFlags,       /* Address of resource manager options*/
     PMQLONG pXARetCode);  /* Address of response from XA call */
```

ax_unreg - AX_UNREG_EXIT:

AX_UNREG_EXIT provides an ax_unreg exit function to perform before and after ax_unreg processing. Use function identifier MQXF_AXUNREG with exit reasons MQXR_BEFORE and MQXR_AFTER to register the before and after ax_unreg call exit functions.

The interface to this function is:

```
AX_UNREG_EXIT (&ExitParms, &ExitContext, &Rmid, &Flags, &XARetCode);
```

where the parameters are:

ExitParms (MQAXP) - input/output
Exit parameter structure.

ExitContext (MQAXC) - input/output
Exit context structure.

Rmid (MQLONG) - input/output
Resource manager identifier.

Flags (MQLONG) - input/output
Resource manager options.

XARetCode (MQLONG) - input/output
Response from XA call.

C language invocation

The queue manager logically defines the following variables:

```
MQAXP  ExitParms;   /* Exit parameter structure */
MQAXC  ExitContext; /* Exit context structure */
MQLONG Rmid;        /* Resource manager identifier */
MQLONG Flags;       /* Resource manager options*/
MQLONG XARetCode;   /* Response from XA call */
```

The queue manager then logically calls the exit as follows:

```
AX_UNREG_EXIT (&ExitParms, &ExitContext, &Rmid, &Flags, &XARetCode);
```

Your exit must match the following C function prototype:

```
typedef void MQENTRY AX_UNREG_EXIT (
     PMQAXP  pExitParms,   /* Address of exit parameter structure */
     PMQAXC  pExitContext, /* Address of exit context structure */
     PMQLONG pRmid,        /* Address of resource manager identifier */
     PMQLONG pFlags,       /* Address of resource manager options*/
     PMQLONG pXARetCode);  /* Address of response from XA call */
```

General information on invoking exit functions:

This topic provides some general guidance to help you to plan your exits, particularly related to handling errors and unexpected events.

Exit failure:

If an exit function abnormally terminates after a destructive, out of syncpoint, MQGET call but before the message has been passed to the application, the exit handler can recover from the failure, and pass control to the application.

In this case, the message might be lost. This is like what happens when an application fails immediately after receiving a message from a queue.

The MQGET call might complete with MQCC_FAILED and MQRC_API_EXIT_ERROR.

If a *before* API call exit function terminates abnormally, the exit handler can recover from the failure and pass control to the application without processing the API call. In this event, the exit function must recover any resources that it owns.

If chained exits are in use, the *after* API call exits for any *before* API call exits that had successfully been driven can themselves be driven. The API call might fail with MQCC_FAILED and MQRC_API_EXIT_ERROR.

Example error handling for exit functions:

The following diagram shows the points (e*N*) at which errors can occur. It is only an example to show how exits behave and should be read together with the following table. In this example, two exit functions are invoked both before and after each API call to show the behavior with chained exits.

```
Application ErrPt     Exit function            API call
----------- -----     -------------            --------

Start

MQCONN  -->
            e1
                           MQ_INIT_EXIT
            e2
                  before MQ_CONNX_EXIT  1
            e3
                  before MQ_CONNX_EXIT  2
            e4
                                          -->  MQCONN
            e5
                  after  MQ_CONNX_EXIT  2
            e6
                  after  MQ_CONNX_EXIT  1
            e7
        <--
MQOPEN  -->
                  before MQ_OPEN_EXIT  1
            e8
                  before MQ_OPEN_EXIT  2
            e9
                                          -->  MQOPEN
            e10
                  after  MQ_OPEN_EXIT  2
            e11
                  after  MQ_OPEN_EXIT  1
            e12
        <--
MQPUT   -->
                  before MQ_PUT_EXIT   1
            e13
                  before MQ_PUT_EXIT   2
            e14
                                          -->  MQPUT
```

```
            e15
                  after  MQ_PUT_EXIT   2
            e16
                  after  MQ_PUT_EXIT   1
            e17
        <--
MQCLOSE -->
                  before MQ_CLOSE_EXIT 1
            e18
                  before MQ_CLOSE_EXIT 2
            e19
                                       -->  MQCLOSE
            e20
                  after  MQ_CLOSE_EXIT 2
            e21
                  after  MQ_CLOSE_EXIT 1
            e22
        <--
MQDISC  -->
                  before MQ_DISC_EXIT  1
            e23
                  before MQ_DISC_EXIT  2
            e24
                                       -->  MQDISC
            e25
                  after  MQ_DISC_EXIT  2
            e26
                  after  MQ_DISC_EXIT  1
            e27

        <--

end
```

The following table lists the actions to be taken at each error point. Only a subset of the error points have been covered, as the rules shown here can apply to all others. It is the actions that specify the intended behavior in each case.

Table 241. API exit errors and appropriate actions to take

ErrPt	Description	Actions
e1	Error while setting up environment setup.	1. Undo environment setup as required 2. Drive no exit functions 3. Fail MQCONN with MQCC_FAILED, MQRC_API_EXIT_LOAD_ERROR
e2	MQ_INIT_EXIT function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Clean up environment 2. Fail MQCONN with MQCC_FAILED, MQRC_API_EXIT_INIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_*[1] 2. Clean up environment

Table 241. API exit errors and appropriate actions to take (continued)

ErrPt	Description	Actions
e3	*Before* MQ_CONNX_EXIT 1 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive MQ_TERM_EXIT function 2. Clean up environment 3. Fail MQCONN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_*[1] 2. Drive MQ_TERM_EXIT function if required 3. Clean up environment if required
e4	*Before* MQ_CONNX_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_CONNX_EXIT 1 function 2. Drive MQ_TERM_EXIT function 3. Clean up environment 4. Fail MQCONN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_*[1] 2. Drive *after* MQ_CONNX_EXIT 1 function if exit not suppressed 3. Drive MQ_TERM_EXIT function if required 4. Clean up environment if required
e5	MQCONN call fails.	1. Pass MQCONN CompCode and Reason 2. Drive *after* MQ_CONNX_EXIT 2 function if the *before* MQ_CONNX_EXIT 2 succeeded and the exit is not suppressed 3. Drive *after* MQ_CONNX_EXIT 1 function if the *before* MQ_CONNX_EXIT 1 succeeded and the exit is not suppressed 4. Drive MQ_TERM_EXIT function 5. Clean up environment
e6	*After* MQ_CONNX_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_CONNX_EXIT 1 function 2. Complete MQCONN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_*[1] 2. Drive *after* MQ_CONNX_EXIT 1 function if required
e7	*After* MQ_CONNX_EXIT 1 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED, complete MQCONN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_*, act as for the values of MQXCC_* and MQXR2_*[1]
e8	*Before* MQ_OPEN_EXIT 1 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED, complete MQOPEN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_*, act as for the values of MQXCC_* and MQXR2_*[1]

Table 241. API exit errors and appropriate actions to take (continued)

ErrPt	Description	Actions
e9	*Before* MQ_OPEN_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_OPEN_EXIT 1 function 2. Complete MQOPEN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_*, act as for the values of MQXCC_* and MQXR2_*[1]
e10	MQOPEN call fails	1. Pass MQOPEN CompCode and Reason 2. Drive *after* MQ_OPEN_EXIT 2 function if exit not suppressed 3. Drive *after* MQ_OPEN_EXIT 1 function if exit not suppressed and if chained exits not suppressed
e11	*After* MQ_OPEN_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_OPEN_EXIT 1 function 2. Complete MQOPEN call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_*[1] 2. Drive *after* MQ_OPEN_EXIT 1 function if exit not suppressed
e25	*After* MQ_DISC_EXIT 2 function completes with: • MQXCC_FAILED • MQXCC_*	• For MQXCC_FAILED: 1. Drive *after* MQ_DISC_EXIT 1 function 2. Drive MQ_TERM_EXIT function 3. Clean up exit execution environment 4. Complete MQDISC call with MQCC_FAILED, MQRC_API_EXIT_ERROR • For MQXCC_* 1. Act as for the values of MQXCC_* and MQXR2_*[1] 2. Drive MQ_TERM_EXIT function 3. Clean up exit execution environment

Note:

1. The values of MQXCC_* and MQXR2_* and their corresponding actions are defined in How queue managers process exit functions.

ExitResponse fields set incorrectly:

This topic gives information about what would happen when the ExitResponse field is set to anything but the supported values.

If the ExitResponse field is set to a value other than one of the supported values, the following actions apply:

- For a *before* MQCONN or MQDISC API exit function:
  - The ExitResponse2 value is ignored.
  - No further *before* exit functions in the exit chain (if any) are invoked; the API call itself is not issued.
  - For any *before* exits that were successfully called, the *after* exits are called in reverse order.
  - If registered, the termination exit functions for those *before* MQCONN or MQDISC exit functions in the chain that were successfully invoked are driven to clean up after these exit functions.
  - The MQCONN or MQDISC call fails with MQRC_API_EXIT_ERROR.
- For a *before* WebSphere MQ API exit function other than MQCONN or MQDISC:

  - The ExitResponse2 value is ignored.
  - No further *before* or *after* data conversion functions in the exit chain (if any) are invoked.
  - For any *before* exits that were successfully called, the *after* exits are called in reverse order.
  - The WebSphere MQ API call itself is not issued.
  - The WebSphere MQ API call fails with MQRC_API_EXIT_ERROR.
- For an *after* MQCONN or MQDISC API exit function:
  - The ExitResponse2 value is ignored.
  - The remaining exit functions that were successfully called before the API call are called in reverse order.
  - If registered, the termination exit functions for those *before* or *after* MQCONN or MQDISC exit functions in the chain that were successfully invoked are driven to clean up after the exit.
  - A CompCode of the more severe of MQCC_WARNING and the CompCode returned by the exit is returned to the application.
  - A Reason of MQRC_API_EXIT_ERROR is returned to the application.
  - The WebSphere MQ API call is successfully issued.
- For an *after* WebSphere MQ API call exit function other than MQCONN or MQDISC:
  - The ExitResponse2 value is ignored.
  - The remaining exit functions that were successfully called before the API call are called in reverse order.
  - A CompCode of the more severe of MQCC_WARNING and the CompCode returned by the exit is returned to the application.
  - A Reason of MQRC_API_EXIT_ERROR is returned to the application.
  - The WebSphere MQ API call is successfully issued.
- For the *before* data conversion on get exit function:
  - The ExitResponse2 value is ignored.
  - The remaining exit functions that were successfully called before the API call are called in reverse order.
  - The message is not converted, and the unconverted message is returned to the application.
  - A CompCode of the more severe of MQCC_WARNING and the CompCode returned by the exit is returned to the application.
  - A Reason of MQRC_API_EXIT_ERROR is returned to the application.
  - The WebSphere MQ API call is successfully issued.

  Note: As the error is with the exit, it is better to return MQRC_API_EXIT_ERROR than to return MQRC_NOT_CONVERTED.

If an exit function sets the ExitResponse2 field to a value other than one of the supported values, a value of MQXR2_DEFAULT_CONTINUATION is assumed instead.

Installable services interface reference information

This collection of topics provides reference information for the installable services.

The functions and data types are listed in alphabetical order within the group for each service type.

How the functions are shown:

How the installable services functions are documented.

For each function there is a description, including the function identifier (for MQZEP).

The *parameters* are shown listed in the order they must occur. They must all be present.

Each parameter name is followed by its data type. These are the elementary data types described in the "Elementary data types" on page 1530.

The C language invocation is also given, after the description of the parameters.

MQZ_AUTHENTICATE_USER - Authenticate user:

This function is provided by an MQZAS_VERSION_5 authorization service component, and is invoked by the queue manager to authenticate a user, or to set identity context fields. It is invoked when WebSphere MQ's user application context is established.

The application context is established during connect calls at the point where the application's user context is initialized, and at each point where the application's user context is changed. Each time a connect call is made, the application's user context information is reacquired in the *IdentityContext* field.

The function identifier for this function (for MQZEP) is MQZID_AUTHENTICATE_USER.

Syntax

MQZ_AUTHENTICATE_USER (*QMgrName*, *SecurityParms*, *ApplicationContext*, *IdentityContext*, *CorrelationPtr*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

SecurityParms
Type: MQCSP - input

Security parameters. Data relating to the user ID, password, and authentication type. If the AuthenticationType attribute of the MQCSP structure is specified as MQCSP_AUTH_USER_ID_AND_PWD, both the user ID and password are compared against the equivalent fields in the IdentityContext (MQZIC) parameter to determine whether they match. For more information, see "MQCSP - Security parameters" on page 1621.

During an MQCONN MQI call this parameter contains null, or default values.

ApplicationContext
Type: MQZAC - input

Application context. Data relating to the calling application. See MQZAC - Application context for details.

During every MQCONN or MQCONNX MQI call, the user context information in the MQZAC structure is reacquired.

IdentityContext
Type: MQZIC - input/output

Identity context. On input to the authenticate user function, this identifies the current identity context. The authenticate user function can change this, at which point the queue manager adopts the new identity context. See MQZIC - Identity context for more details on the MQZIC structure.

CorrelationPtr
Type: MQPTR - output

Correlation pointer. Specifies the address of any correlation data. This pointer is subsequently passed on to other OAM calls.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation flag. You can specify the following values:

MQZCI_DEFAULT
Continuation dependent on other components.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information on these reason codes, see Reason codes.

C invocation

```
MQZ_AUTHENTICATE_USER (QMgrName, SecurityParms, ApplicationContext,
                  IdentityContext, &CorrelationPtr, ComponentData,
                  &Continuation, &CompCode, &Reason);
```

Declare the parameters passed to the service as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */
MQCSP     SecurityParms;      /* Security parameters */
MQZAC     ApplicationContext; /* Application context */
MQZIC     IdentityContext;    /* Identity context */
MQPTR     CorrelationPtr;     /* Correlation pointer */
MQBYTE    ComponentData[n];   /* Component data */
MQLONG    Continuation;       /* Continuation indicator set by
                                 component */
MQLONG    CompCode;           /* Completion code */
MQLONG    Reason;             /* Reason code qualifying CompCode */
```

MQZ_CHECK_AUTHORITY - Check authority:

This function is provided by a MQZAS_VERSION_1 authorization service component, and is started by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

Syntax

MQZ_CHECK_AUTHORITY(*QMgrName*, *EntityName*, *EntityType*, *ObjectName*, *ObjectType*, *Authority*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityName
Type: MQCHAR12 - input

Entity name. The name of the entity whose authorization to the object is to be checked. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by EntityName. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

Authority
Type: MQLONG - input

Authority to be checked. If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT
Ability to use the MQCONN call.

MQZAO_BROWSE
Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT
Principal. Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT
Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE
Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET
Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT
Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT
Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT
Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT
Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY
Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI
All of the MQI authorizations.

This enables all of the authorizations.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE
Ability to create objects of a specified type.

MQZAO_DELETE
Ability to delete a specified object.

MQZAO_DISPLAY
Ability to display the attributes of a specified object.

MQZAO_CHANGE
Ability to change the attributes of a specified object.

MQZAO_CLEAR
Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE
Ability to authorize other users for a specified object.

MQZAO_CONTROL
Ability to start or stop a listener, service, or non-client channel object, and the ability to ping a non-client channel object.

MQZAO_CONTROL_EXTENDED
Ability to reset a sequence number, or resolve an indoubt message on a non-client channel object.

MQZAO_ALL_ADMIN
Ability to set identity context.

All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL
All authorizations, other than MQZAO_CREATE.

MQZAO_NONE
No authorizations.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

If the call to a component fails (that is, *CompCode* returns MQCC_FAILED), and the *Continuation* parameter is MQZCI_DEFAULT or MQZCI_CONTINUE, the queue manager continues to call other components if there are any.

If the call succeeds (that is, *CompCode* returns MQCC_OK) no other components are called no matter what the setting of *Continuation* is.

If the call fails and the *Continuation* parameter is MQZCI_STOP then no other components are called and the error is returned to the queue manager. Components have no knowledge of previous calls, so the *Continuation* parameter is always set to MQZCI_DEFAULT before the call.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_CHECK_AUTHORITY_2 - Check authority (extended):

This function is provided by a MQZAS_VERSION_2 authorization service component, and is started by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

MQZ_CHECK_AUTHORITY_2 is like MQZ_CHECK_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

MQZ_CHECK_AUTHORITY_2(*QMgrName*, *EntityData*, *EntityType*, *ObjectName*, *ObjectType*, *Authority*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity with authorization to the object that is to be checked. See "MQZED - Entity descriptor" on page 2537 for details.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityData*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority to be checked. If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT
Ability to use the MQCONN call.

MQZAO_BROWSE
Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT
Principal. Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT
Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE
Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET
Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT
Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT
Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT
Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT
Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY
Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI
All of the MQI authorizations.

This enables all of the authorizations.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE
Ability to create objects of a specified type.

MQZAO_DELETE
Ability to delete a specified object.

MQZAO_DISPLAY
Ability to display the attributes of a specified object.

MQZAO_CHANGE
Ability to change the attributes of a specified object.

MQZAO_CLEAR
Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE
Ability to authorize other users for a specified object.

MQZAO_CONTROL
Ability to start or stop a listener, service, or non-client channel object, and the ability to ping a non-client channel object.

MQZAO_CONTROL_EXTENDED
Ability to reset a sequence number, or resolve an indoubt message on a non-client channel object.

MQZAO_ALL_ADMIN
Ability to set identity context.

All of the administration authorizations, other than MQZAO_CREATE.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_ALL
All authorizations, other than MQZAO_CREATE.

MQZAO_NONE
No authorizations.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_CHECK_AUTHORITY_2 (QMgrName, &EntityData, EntityType,
                       ObjectName, ObjectType, Authority, ComponentData,
                       &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_CHECK_PRIVILEGED - Check if user is privileged:

This function is provided by an MQZAS_VERSION_6 authorization service component, and is invoked by the queue manager to determine whether a specified user is a privileged user.

The function identifier for this function (for MQZEP) is MQZID_CHECK_PRIVILEGED.

Syntax

```
MQZ_CHECK_PRIVILEGED( QMgrName, EntityData, EntityType, ComponentData, Continuation,
 CompCode, Reason)
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity that is to be checked. For more information, see "MQZED - Entity descriptor" on page 2537.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by EntityData. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

If the call to a component fails (that is, *CompCode* returns MQCC_FAILED), and the *Continuation* parameter is MQZCI_DEFAULT or MQZCI_CONTINUE, the queue manager continues to call other components if there are any.

If the call succeeds (that is, *CompCode* returns MQCC_OK) no other components are called no matter what the setting of *Continuation* is.

If the call fails and the *Continuation* parameter is MQZCI_STOP then no other components are called and the error is returned to the queue manager. Components have no knowledge of previous calls, so the *Continuation* parameter is always set to MQZCI_DEFAULT before the call.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_PRIVILEGED
(2584, X'A18') This user is not a privileged user ID.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_CHECK_PRIVILEGED (QMgrName, &EntityData, EntityType,
                      ComponentData, &Continuation,
                      &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_COPY_ALL_AUTHORITY - Copy all authority:

This function is provided by an authorization service component. It is started by the queue manager to copy all of the authorizations that are currently in force for a reference object to another object.

The function identifier for this function (for MQZEP) is MQZID_COPY_ALL_AUTHORITY.

Syntax

MQZ_COPY_ALL_AUTHORITY(*QMgrName*, *RefObjectName*, *ObjectName*, *ObjectType*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

RefObjectName
Type: MQCHAR48 - input

Reference object name. The name of the reference object, the authorizations for which are to be copied. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which accesses are to be set. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *RefObjectName* and *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_REF_OBJECT
(2294, X'8F6') Reference object unknown.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName, ObjectType,
                        ComponentData, &Continuation, &CompCode,
                        &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  RefObjectName;     /* Reference object name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_DELETE_AUTHORITY - Delete authority:

This function is provided by an authorization service component, and is started by the queue manager to delete all of the authorizations associated with the specified object.

The function identifier for this function (for MQZEP) is MQZID_DELETE_AUTHORITY.

Syntax

MQZ_DELETE_AUTHORITY(*QMgrName*, *ObjectName*, *ObjectType*,*ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which accesses are to be deleted. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_DELETE_AUTHORITY (QMgrName, ObjectName, ObjectType, ComponentData,
                      &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_ENUMERATE_AUTHORITY_DATA - Enumerate authority data:

This function is provided by an MQZAS_VERSION_4 authorization service component, and is started repeatedly by the queue manager to retrieve all of the authority data that matches the selection criteria specified on the first invocation.

The function identifier for this function (for MQZEP) is MQZID_ENUMERATE_AUTHORITY_DATA.

Syntax

```
MQZ_ENUMERATE_AUTHORITY_DATA(QMgrName, StartEnumeration, Filter, AuthorityBufferLength,
AuthorityBuffer, AuthorityDataLength, ComponentData, Continuation, CompCode, Reason)
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

StartEnumeration
Type: MQLONG - input

Flag indicating whether call can start enumeration. This indicates whether the call can start the enumeration of authority data, or continue the enumeration of authority data started by a previous call to MQZ_ENUMERATE_AUTHORITY_DATA. The value is one of the following values:

MQZSE_START
Start enumeration. The call is started with this value to start the enumeration of authority data. The *Filter* parameter specifies the selection criteria to be used to select the authority data returned by this and successive calls.

MQZSE_CONTINUE
Continue enumeration. The call is started with this value to continue the enumeration of authority data. The *Filter* parameter is ignored in this case, and can be specified as the null pointer (the selection criteria are determined by the *Filter*parameter specified by the call that had *StartEnumeration* set to MQZSE_START).

Filter
Type: MQZAD - input

Filter. If *StartEnumeration* is MQZSE_START, *Filter* specifies the selection criteria to be used to select the authority data to return. If *Filter* is the null pointer, no selection criteria are used, that is, all authority data is returned. See "MQZAD - Authority data" on page 2534 for details of the selection criteria that can be used.

If *StartEnumeration* is MQZSE_CONTINUE, *Filter* is ignored, and can be specified as the null pointer.

AuthorityBufferLength
Type: MQLONG - input

Length of *AuthorityBuffer*. This is the length in bytes of the *AuthorityBuffer* parameter. The authority buffer must be large enough to accommodate the data to be returned.

AuthorityBuffer
Type: MQZAD - output

Authority data. This is the buffer in which the authority data is returned. The buffer must be large enough to accommodate an MQZAD structure, an MQZED structure, plus the longest entity name and longest domain name defined.

Note: Note: This parameter is defined as an MQZAD, as the MQZAD always occurs at the start of the buffer. However, if the buffer is declared as an MQZAD, the buffer will be too small - it must be bigger than an MQZAD so that it can accommodate the MQZAD, MQZED, plus entity and domain names.

AuthorityDataLength
Type: MQLONG - output

Length of data returned in *AuthorityBuffer*. If the authority buffer is too small, *AuthorityDataLength* is set to the length of the buffer required, and the call returns completion code MQCC_FAILED and reason code MQRC_BUFFER_LENGTH_ERROR.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_ENUMERATE_AUTHORITY_DATA, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_BUFFER_LENGTH_ERROR
(2005, X'7D5') Buffer length parameter not valid.

MQRC_NO_DATA_AVAILABLE
(2379, X'94B') No data available.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_ENUMERATE_AUTHORITY_DATA (QMgrName, StartEnumeration, &Filter,
                              AuthorityBufferLength,
                              &AuthorityBuffer,
                              &AuthorityDataLength, ComponentData,
                              &Continuation, &CompCode,
                              &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;               /* Queue manager name */
MQLONG    StartEnumeration;       /* Flag indicating whether call should
                                     start enumeration */
MQZAD     Filter;                 /* Filter */
MQLONG    AuthorityBufferLength;  /* Length of AuthorityBuffer */
MQZAD     AuthorityBuffer;        /* Authority data */
MQLONG    AuthorityDataLength;    /* Length of data returned in
                                     AuthorityBuffer */
MQBYTE    ComponentData[n];       /* Component data */
```

```
MQLONG    Continuation;          /* Continuation indicator set by
                                    component */
MQLONG    CompCode;              /* Completion code */
MQLONG    Reason;                /* Reason code qualifying CompCode */
```

MQZ_FREE_USER - Free user:

This function is provided by a MQZAS_VERSION_5 authorization service component, and is started by the queue manager to free associated allocated resource.

It is started when an application has finished running under all user contexts, for example during an MQDISC MQI call.

The function identifier for this function (for MQZEP) is MQZID_FREE_USER.

Syntax

MQZ_FREE_USER(*QMgrName*, *FreeParms*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

FreeParms
Type: MQZFP - input

Free parameters. A structure containing data relating to the resource to be freed. See "MQZFP - Free parameters" on page 2540 for details.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation flag. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on other components.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_AUTHENTICATE_USER (QMgrName, SecurityParms, ApplicationContext,
                   IdentityContext, CorrelationPtr, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;           /* Queue manager name */
MQZFP     FreeParms;          /* Resource to be freed */
MQBYTE    ComponentData[n];   /* Component data */
MQLONG    Continuation;       /* Continuation indicator set by
                                 component */
MQLONG    CompCode;           /* Completion code */
MQLONG    Reason;             /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY - Get authority:

This function is provided by a MQZAS_VERSION_1 authorization service component, and is started by the queue manager to retrieve the authority that an entity has to access the specified object, including (if the entity is a principal) authorities possessed by the groups in which the principal is a member. Authorities from generic profiles are included in the returned authority set.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

Syntax

```
MQZ_GET_AUTHORITY( QMgrName, EntityName, EntityType, ObjectName, ObjectType, Authority,
 ComponentData, Continuation, CompCode, Reason)
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityName
Type: MQCHAR12 - input

Entity name. The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityName*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which access is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, &Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY_2 - Get authority (extended):

This function is provided by a MQZAS_VERSION_2 authorization service component, and is started by the queue manager to retrieve the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

MQZ_GET_AUTHORITY_2 is like MQZ_GET_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

MQZ_GET_AUTHORITY_2(*QMgrName*, *EntityData*, *EntityType*, *ObjectName*, *ObjectType*, *Authority*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity for which authorization to the object is to be retrieved. See "MQZED - Entity descriptor" on page 2537 for details.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityData*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which the entity authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode

Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

Syntax

```
MQZ_GET_AUTHORITY_2(QMgrName, EntityData, EntityType, ObjectName, ObjectType, Authority,
ComponentData, Continuation, CompCode, Reason)
```

C invocation

```
MQZ_GET_AUTHORITY_2 (QMgrName, &EntityData, EntityType, ObjectName,
                     ObjectType, &Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY - Get explicit authority:

This function is provided by a MQZAS_VERSION_1 authorization service component, and is started by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

On UNIX platforms, for the built-in WebSphere MQ object authority manager (OAM), the returned authority is that possessed only by the principal's primary group.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

Syntax

```
MQZ_GET_EXPLICIT_AUTHORITY( QMgrName, EntityName, EntityType, ObjectName, ObjectType,
 Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityName
Type: MQCHAR12 - input

Entity name. The name of the entity for which access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityName*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which the entity authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
                            ObjectName, ObjectType, &Authority,
                            ComponentData, &Continuation,
                            &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY_2 - Get explicit authority (extended):

This function is provided by a MQZAS_VERSION_2 authorization service component, and is started by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

MQZ_GET_EXPLICIT_AUTHORITY_2 is like MQZ_GET_EXPLICIT_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

```
MQZ_GET_EXPLICIT_AUTHORITY_2(QMgrName, EntityData, EntityType, ObjectName, ObjectType,
Authority, ComponentData, Continuation, CompCode, Reason)
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity whose authorization to the object is to be retrieved. See "MQZED - Entity descriptor" on page 2537 for details.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityData*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object for which the entity authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY_2 (QMgrName, &EntityData, EntityType,
                              ObjectName, ObjectType, &Authority,
                              ComponentData, &Continuation,
                              &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_AUTHORITY - Initialize authorization service:

This function is provided by an authorization service component, and is started by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_AUTHORITY.

Syntax

MQZ_INIT_AUTHORITY(*Hconfig*, *Options*, *QMgrName*, *ComponentDataLength*, *ComponentData*, *Version*, *CompCode*, *Reason*)

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options
Type: MQLONG - input

Initialization options. It must be one of the following values:

MQZIO_PRIMARY
Primary initialization.

MQZIO_SECONDARY
Secondary initialization.

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentDataLength
Type: MQLONG - input

Length of component data. Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This is initialized to all zeros before calling the component primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Version
Type: MQLONG - input/output

Version number. On input to the initialization function, this identifies the highest version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which it supports. If on return the queue manager does not support the version returned by the component, it calls the component MQZ_TERM_AUTHORITY function and makes no further use of this component.

The following values are supported:

MQZAS_VERSION_1
Version 1.

MQZAS_VERSION_2
Version 2.

MQZAS_VERSION_3
Version 3.

MQZAS_VERSION_4
Version 4.

MQZAS_VERSION_5
Version 5.

MQZAS_VERSION_6
Version 6.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_INIT_AUTHORITY (Hconfig, Options, QMgrName, ComponentDataLength,
                    ComponentData, &Version, &CompCode,
                    &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_INQUIRE - Inquire authorization service:

This function is provided by a MQZAS_VERSION_5 authorization service component, and is started by the queue manager to query the supported functionality.

Where multiple service components are used, service components are called in reverse order to the order they were installed in.

The function identifier for this function (for MQZEP) is MQZID_INQUIRE.

Syntax

MQZ_INQUIRE(*QMgrName*, *SelectorCount*, *Selectors*, *IntAttrCount*, *IntAttrs*, *CharAttrLength*, *CharAttrs*, *SelectorReturned*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

SelectorCount
Type: MQLONG - input

Number of selectors. The number of selectors supplied in the *Selectors* parameter.

The value must be in the range 0 through 256.

Selectors
Type: MQLONGxSelectorCount - input

Array of selectors. Each selector identifies a required attribute and must be one of the following:

- MQIACF_INTERFACE_VERSION (integer)
- MQIACF_USER_ID_SUPPORT (integer)
- MQCACF_SERVICE_COMPONENT (character)

Selectors can be specified in any order. The number of selectors in the array is indicated by the *SelectorCount* parameter.

Integer attributes identified by selectors are returned in the *IntAttrs* parameter in the same order as they appear in *Selectors* .

Character attributes identified by selectors are returned in the *CharAttrs* parameter in the same order as they in appear *Selectors*.

IntAttrCount
Type: MQLONG - input

Number of integer attributes supplied in the IntAttrs parameter.

The value must be in the range 0 through 256.

IntAttrs
Type: MQLONG×IntAttrCount - output

Integer attributes. Array of integer attributes. The integer attributes are returned in the same order as the corresponding integer selectors in the *Selectors* array.

CharAttrCount
Type: MQLONG - input

Length of the character attributes buffer. The length in bytes of the *CharAttrs* parameter.

The value must be at least the sum of the lengths of the requested character attributes. If no character attributes are requested, zero is a valid value.

CharAttrs
Type: MQLONG×CharAttrCount - output

Character attributes buffer. Buffer containing character attributes, concatenated together. The character attributes are returned in the same order as the corresponding character selectors in the *Selectors* array.

The length of the buffer is given by the CharAttrCount parameter.

SelectorReturned
Type: MQLONG×SelectorCount - input

Selector returned. Array of values identifying which attributes have been returned from the set requested for by the selectors in the Selectors parameter. The number of values in this array is indicated by the *SelectorCount* parameter. Each value in the array relates to the selector from the corresponding position in the Selectors array. Each value is one of the following:

MQZSL_RETURNED
The attribute requested by the corresponding selector in the *Selectors* parameter has been returned.

MQZSL_NOT_RETURNED
The attribute requested by the corresponding selector in the *Selectors* parameter has not been returned.

The array is initialized with all values as *MQZSL_NOT_RETURNED*. When an authorization service component returns an attribute, it sets the appropriate value in the array to *MQZSL_NOT_RETURNED*. This allows any other authorization service components, to which the inquire call is made, to identify which attributes have already been returned.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_WARNING
Partial completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode* .

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_CHAR_ATTRS_TOO_SHORT
Not enough space for character attributes.

MQRC_INT_COUNT_TOO_SMALL
Not enough space for integer attributes.

If *CompCode* is MQCC_FAILED:

MQRC_SELECTOR_COUNT_ERROR
Number of selectors is not valid.

MQRC_SELECTOR_ERROR
Attribute selector not valid.

MQRC_SELECTOR_LIMIT_EXCEEDED
Too many selectors specified.

MQRC_INT_ATTR_COUNT_ERROR
Number of integer attributes is not valid.

MQRC_INT_ATTRS_ARRAY_ERROR
Integer attributes array not valid.

MQRC_CHAR_ATTR_LENGTH_ERROR
Number of character attributes is not valid.

MQRC_CHAR_ATTRS_ERROR
Character attributes string is not valid.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_INQUIRE (QMgrName, SelectorCount, Selectors, IntAttrCount,
                  &IntAttrs, CharAttrLength, &CharAttrs,
                  SelectorReturned, ComponentData, &Continuation,
                  &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48   QMgrName;              /* Queue manager name */
MQLONG     SelectorCount;         /* Selector count */
MQLONG     Selectors[n];          /* Selectors */
MQLONG     IntAttrCount;          /* IntAttrs count */
MQLONG     IntAttrs[n];           /* Integer attributes */
MQLONG     CharAttrCount;         /* CharAttrs count */
MQLONG     CharAttrs[n];          /* Chatacter attributes */
MQLONG     SelectorReturned[n];   /* Selector returned */
MQBYTE     ComponentData[n];      /* Component data */
MQLONG     Continuation;          /* Continuation indicator set by
                                     component */
MQLONG     CompCode;              /* Completion code */
MQLONG     Reason;                /* Reason code qualifying CompCode */
```

MQZ_REFRESH_CACHE - Refresh all authorizations:

This function is provided by an MQZAS_VERSION_3 authorization service component, and is invoked by the queue manager to refresh the list of authorizations held internally by the component.

The function identifier for this function (for MQZEP) is MQZID_REFRESH_CACHE (8L).

Syntax

```
MQZ_REFRESH_CACHE(QMgrName, ComponentData, Continuation, CompCode, Reason)
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

C invocation

```
MQZ_REFRESH_CACHE (QMgrName, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY - Set authority:

This function is provided by a MQZAS_VERSION_1 authorization service component, and is started by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

Syntax

MQZ_SET_AUTHORITY(*QMgrName*, *EntityName*, *EntityType*, *ObjectName*, *ObjectType*, *Authority*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityName
Type: MQCHAR12 - input

Entity name. The name of the entity for which access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityName*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If one authority is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authority is being set, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY, this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
```

```
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY_2 - Set authority (extended):

This function is provided by a MQZAS_VERSION_2 authorization service component, and is started by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

MQZ_SET_AUTHORITY_2 is like MQZ_SET_AUTHORITY, but with the *EntityName* parameter replaced by the *EntityData* parameter.

Syntax

MQZ_SET_AUTHORITY_2(*QMgrName*, *EntityData*, *EntityType*, *ObjectName*, *ObjectType*, *Authority*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

EntityData
Type: MQZED - input

Entity data. Data relating to the entity whose authorization to the object is to be set. See "MQZED - Entity descriptor" on page 2537 for details.

EntityType
Type: MQLONG - input

Entity type. The type of entity specified by *EntityData*. It must be one of the following values:

MQZAET_PRINCIPAL
Principal.

MQZAET_GROUP
Group.

ObjectName
Type: MQCHAR48 - input

Object name. The name of the object to which the entity authority is to be set. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType
Type: MQLONG - input

Object type. The type of entity specified by *ObjectName*. It must be one of the following values:

MQOT_AUTH_INFO
Authentication information.

MQOT_CHANNEL
Channel.

MQOT_CLNTCONN_CHANNEL
Client connection channel.

MQOT_LISTENER
Listener.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process definition.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_SERVICE
Service.

MQOT_TOPIC
Topic.

Authority
Type: MQLONG - input

Authority of entity. If one authority is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authority is being set, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY, this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_SET_AUTHORITY_2 (QMgrName, &EntityData, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQZED     EntityData;        /* Entity data */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_AUTHORITY - Terminate authorization service:

This function is provided by an authorization service component, and is started by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_AUTHORITY.

Syntax

```
MQZ_TERM_AUTHORITY(Hconfig, Options, QMgrName, ComponentData, CompCode, Reason)
```

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the particular component being terminated. It is to be used by the component when calling the queue manager with the MQZEP function.

Options
Type: MQLONG - input

Termination options. It must be one of the following values:

MQZTO_PRIMARY
Primary termination.

MQZTO_SECONDARY
Secondary termination.

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter on the MQZ_INIT_AUTHORITY call.

When the MQZ_TERM_AUTHORITY call has completed, the queue manager discards this data.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData,
                    &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZ_DELETE_NAME - Delete name:

This function is provided by a name service component, and is started by the queue manager to delete an entry for the specified queue.

The function identifier for this function (for MQZEP) is MQZID_DELETE_NAME.

Syntax

MQZ_DELETE_NAME(*QMgrName*, *QName*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

QName
Type: MQCHAR48 - input

Queue name. The name of the queue for which an entry is to be deleted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the ComponentDataLength parameter on the MQZ_INIT_NAME call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. It must be one of the following values:

MQZCI_DEFAULT
Continuation dependent on queue manager.

MQZCI_STOP
Do not continue with next component.

For the **MQZ_DELETE_NAME** command, the queue manager does not attempt to start another component, no matter what is returned in the **Continuation** parameter.

CompCode

Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason

Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_UNKNOWN_NAME
(2288, X'8F0') Queue name not found.

Note: It might not be possible to return this code if the underlying service responds with success for this case.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_DELETE_NAME (QMgrName, QName, ComponentData, &Continuation,
                 &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_NAME - Initialize name service:

This function is provided by a name service component, and is started by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_NAME.

Syntax

MQZ_INIT_NAME(*Hconfig*, *Options*, *QMgrName*, *ComponentDataLength*, *ComponentData*, *Version*, *CompCode*, *Reason*)

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options
Type: MQLONG - input

Initialization options. It must be one of the following values:

MQZIO_PRIMARY
Primary initialization.

MQZIO_SECONDARY
Secondary initialization.

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentDataLength
Type: MQLONG - input

Length of component data. Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This is initialized to all zeros before calling the component primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Version
Type: MQLONG - input/output

Version number. On input to the initialization function, this identifies the highest version number that the queue manager supports. The initialization function must change this, if necessary, to the version

of the interface which it supports. If on return the queue manager does not support the version returned by the component, it calls the component MQZ_TERM_NAME function and makes no further use of this component.

The following values are supported:

MQZAS_VERSION_1
Version 1.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_INIT_NAME (Hconfig, Options, QMgrName, ComponentDataLength,
               ComponentData, &Version, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_INSERT_NAME - Insert name:

This function is provided by a name service component, and is started by the queue manager to insert an entry for the specified queue, containing the name of the queue manager that owns the queue. If the queue is already defined in the service, the call fails.

The function identifier for this function (for MQZEP) is MQZID_INSERT_NAME.

Syntax

```
MQZ_INSERT_NAME(QMgrName, QName, ResolvedQMgrName, ComponentData, Continuation, CompCode,
Reason)
```

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

QName
Type: MQCHAR48 - input

Queue name. The name of the queue for which an entry is to be inserted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName
Type: MQCHAR48 - input

Resolved queue manager name. The name of the queue manager to which the queue resolves. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation
Type: MQLONG - input/output

Continuation indicator set by component. For MQZ_INSERT_NAME, the queue manager does not attempt to start another component, whatever is returned in the *Continuation* parameter.

The following values are supported:

MQZCI_DEFAULT
Continuation dependent on queue manager.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_Q_ALREADY_EXISTS
(2290, X'8F2') Queue object already exists.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQCHAR48  ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_LOOKUP_NAME - Lookup name:

This function is provided by a name service component, and is started by the queue manager to retrieve the name of the owning queue manager, for a specified queue.

The function identifier for this function (for MQZEP) is MQZID_LOOKUP_NAME.

Syntax

MQZ_LOOKUP_NAME(*QMgrName*, *QName*, *ResolvedQMgrName*, *ComponentData*, *Continuation*, *CompCode*, *Reason*)

Parameters

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

QName
Type: MQCHAR48 - input

Queue name. The name of the queue for which an entry is to be resolved. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName
Type: MQCHAR48 - output

Resolved queue manager name. If the function completes successfully, this is the name of the queue manager that owns the queue.

The name returned by the service component must be padded on the right with blanks to the full length of the parameter; the name must not be terminated by a null character, or contain leading or embedded blanks.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation
Type: MQLONG - output

Continuation indicator set by component. For MQZ_LOOKUP_NAME, the queue manager specifies whether to start another name service component, as follows:

- If *CompCode* is MQCC_OK, no further components are started, whatever value is returned in *Continuation*.
- If *CompCode* is not MQCC_OK, a further component is started, unless *Continuation* is MQZCI_STOP.

The following values are supported:

MQZCI_DEFAULT
Continuation dependent on queue manager.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_Q_NAME
(2288, X'8F0') Queue name not found.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_LOOKUP_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

The parameters passed to the service are declared as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQCHAR48  ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_NAME - Terminate name service:

This function is provided by a name service component, and is started by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_NAME.

Syntax

```
MQZ_TERM_NAME(Hconfig, Options, QMgrName, ComponentData, CompCode, Reason)
```

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the particular component being terminated. It is used by the component when calling the queue manager with the MQZEP function.

Options
Type: MQLONG - input

Termination options. It must be one of the following values:

MQZTO_PRIMARY
Primary termination.

MQZTO_SECONDARY
Secondary termination.

QMgrName
Type: MQCHAR48 - input

Queue manager name. The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to use it in any defined manner.

ComponentData
Type: MQBYTE×ComponentDataLength - input/output

Component data. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of these component functions is called.

Component data is in shared memory accessible to all processes.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

When the MQZ_TERM_NAME call has completed, the queue manager discards this data.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZ_TERM_NAME (Hconfig, Options, QMgrName, ComponentData, &CompCode,
               &Reason);
```

The parameters passed to the service are declared as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZAC - Application context:

The MQZAC structure is used on the MQZ_AUTHENTICATE_USER call for the *ApplicationContext* parameter. This parameter specifies data related to the calling application.

Table 1 summarizes the fields in the structure.

Table 242. Fields in MQZAC

Field	Description
StrucId	Structure identifier
Version	Structure version number
ProcessId	Process identifier
ThreadId	Thread identifier
ApplName	Application name
UserID	User identifier
EffectiveUserID	Effective user identifier
Environment	Environment
CallerType	Caller type
AuthenticationType	Authentication type
BindType	Bind type

Fields

StrucId

Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZAC_STRUC_ID
Identifier for application context structure.

For the C programming language, the constant MQZAC_STRUC_ID_ARRAY is also defined; this has the same value as MQZAC_STRUC_ID, but is an array of characters instead of a string.

Version

Type: MQLONG - input

Structure version number. The value is as follows:

MQZAC_VERSION_1
Version-1 application context structure. The constant MQZAC_CURRENT_VERSION specifies the version number of the current version.

ProcessId

Type: MQPID - input

Process identifier of the application.

ThreadId
Type: MQTID - input

Thread identifier of the application.

ApplName
Type: MQCHAR28 - input

Application name.

UserID
Type: MQCHAR12 - input

User identifier. On UNIX systems this field specifies the application's real user ID. On Windows this field specifies the application's user ID.

EffectiveUserID
Type: MQCHAR12 - input

Effective user identifier. On UNIX systems this field specifies the application's effective user ID. On Windows this field is blank.

Environment
Type: MQLONG - input

Environment. This field specifies the environment from which the call was made. The field is one of the following values:

MQXE_COMMAND_SERVER
Command server

MQXE_MQSC
runmqsc command interpreter

MQXE_MCA
Message channel agent MQXE_OTHER

MQXE_OTHER
Undefined environment

CallerType
Type: MQLONG - input

Caller Type. This field specifies the type of program that made the call. The field is one of the following values:

MQXACT_EXTERNAL
The call is external to the queue manager.

MQXACT_INTERNAL
The call is internal to the queue manager.

AuthenticationType
Type: MQLONG - input

Authentication Type. This field specifies the type of authentication being performed. The field is one of the following values:

MQZAT_INITIAL_CONTEXT
The authentication call is due to user context being initialized. This value is used during an MQCONN or MQCONNX call.

MQZAT_CHANGE_CONTEXT
The authentication call is due to the user context being changed. This value is used when the MCA changes the user context. Parent topic: MQZAC -

BindType

Type: MQLONG - input

Bind Type. This field specifies the type of binding in use. The field is one of the following values:

MQCNO_FASTPATH_BINDING
Fastpath binding.

MQCNO_SHARED_BINDING
Shared binding.

MQCNO_ISOLATED_BINDING
Isolated binding.

C declaration

Declare the fields of the structure as follows:

```
typedef struct tagMQZAC MQZAC;
struct tagMQZAC {
  MQCHAR4   StrucId;            /* Structure identifier */
  MQLONG    Version;            /* Structure version number */
  MQPID     ProcessId;          /* Process identifier */
  MQTID     ThreadId;           /* Thread identifier */
  MQCHAR28  ApplName;           /* Application name */
  MQCHAR12  UserID;             /* User identifier */
  MQCHAR12  EffectiveUserID;    /* Effective user identifier */
  MQLONG    Environment;        /* Environment */
  MQLONG    CallerType;         /* Caller type */
  MQLONG    AuthenticationType; /* Authentication type */
  MQLONG    BindType;           /* Bind type */
};
```

MQZAD - Authority data:

The MQZAD structure is used on the MQZ_ENUMERATE_AUTHORITY_DATA call for two parameters, one input and one output.

- MQZAD is used for the *Filter* parameter which is input to the call. This parameter specifies the selection criteria that are to be used to select the authority data returned by the call.
- MQZAD is also used for the *AuthorityBuffer* parameter which is output from the call. This parameter specifies the authorizations for one combination of profile name, object type, and entity.

Table 1. summarizes the fields in the structure.

Table 243. Fields in MQZAD

Field	Description
StrucId	Structure identifier
Version	Structure version number
ProfileName	Process identifier
ObjectType	Thread identifier
Authority	Application name
EntityDataPtr	User identifier
EntityType	Environment
Options	Caller type

Fields

StrucId
Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZAC_STRUC_ID
Identifier for application context structure.

For the C programming language, the constant MQZAC_STRUC_ID_ARRAY is also defined; this has the same value as MQZAC_STRUC_ID, but is an array of characters instead of a string.

Version
Type: MQLONG - input

Structure version number. The value is as follows:

MQZAC_VERSION_1
Version-1 application context structure. The constant MQZAC_CURRENT_VERSION specifies the version number of the current version.

The following constant specifies the version number of the current version:

MQZAD_CURRENT_VERSION
Current version of authority data structure.

ProfileName
Type: MQCHAR48 - input

Profile name.

For the *Filter* parameter, this field is the profile name for which authority data is required. If the name is entirely blank up to the end of the field or the first null character, authority data for all profile names is returned.

For the *AuthorityBuffer* parameter, this field is the name of a profile that matches the specified selection criteria.

ObjectType
Type: MQLONG - input

Object type.

For the *Filter* parameter, this field is the object type for which authority data is required. If the value is MQOT_ALL, authority data for all object types is returned.

For the *AuthorityBuffer* parameter, this field is the object type to which the profile identified by the *ProfileName* parameter applies.

The value is one of the following; for the *Filter* parameter, the value MQOT_ALL is also valid:

MQOT_AUTH_INFO
Authentication information

MQOT_CHANNEL
Channel

MQOT_CLNTCONN_CHANNEL
Client connection channel

MQOT_LISTENER
Listener

MQOT_NAMELIST
Namelist

MQOT_PROCESS
Process definition

MQOT_Q
Queue

MQOT_Q_MGR
Queue manager

MQOT_SERVICE
Service

Authority
Type: MQLONG - input

Authority.

For the *Filter* parameter, this field is ignored.

For the *AuthorityBuffer* parameter, this field represents the authorizations that the entity has to the objects identified by *ProfileName* and *ObjectType*. If the entity has only one authority, the field is equal to the appropriate authorization value (MQZAO_* constant). If the entity has more than one authority, the field is the bitwise OR of the corresponding MQZAO_* constants.

EntityDataPtr
Type: PMQZED - input

Address of MQZED structure identifying an entity.

For the *Filter* parameter, this field points to an MQZED structure that identifies the entity for which authority data is required. If *EntityDataPtr* is the null pointer, authority data for all entities is returned.

For the *AuthorityBuffer* parameter, this field points to an MQZED structure that identifies the entity for which authority data has been returned.

EntityType
Type: MQLONG - input

Entity type.

For the *Filter* parameter, this field specifies the entity type for which authority data is required. If the value is MQZAET_NONE, authority data for all entity types is returned.

For the *AuthorityBuffer* parameter, this field specifies the type of the entity identified by the MQZED structure pointed to by the *EntityDataPtr* parameter.

The value is one of the following; for the *Filter* parameter, the value MQZAET_NONE is also valid:

MQZAET_PRINCIPAL
Principal

MQZAET_GROUP
Group

Options
Type: MQAUTHOPT - input

Options. This field specifies options that give control over the profiles that are displayed. One of the following values must be specified:

MQAUTHOPT_NAME_ALL_MATCHING
Displays all profiles

MQAUTHOPT_NAME_EXPLICIT
Displays profiles that have exactly the same name as specified in the *ProfileName* field.

In addition, one of the following must also be specified:

MQAUTHOPT_ENTITY_SET
Display all profiles that are used to calculate the cumulative authority that the entity has to the object specified by the *ProfileName* parameter. The *ProfileName* parameter must not contain any wildcard characters.

If the specified entity is a principal, for each member of the set {entity, groups} the most applicable profile that applies to the object is displayed.

If the specified entity is a group, the most applicable profile from the group that applies to the object is displayed.

If this value is specified, the values of *ProfileName*, *ObjectType*, *EntityType*, and the entity name that is specified in the *EntityDataPtr* MQZED structure, must all be non-blank.

If you have specified MQAUTHOPT_NAME_ALL_MATCHING, you can also specify the following value:

MQAUTHOPT_ENTITY_EXPLICIT
Displays profiles that have exactly the same entity name as the entity name specified in the *EntityDataPtr* MQZED structure.

C declaration

```
typedef struct tagMQZAD MQZAD;
struct tagMQZAD {
  MQCHAR4   StrucId;        /* Structure identifier */
  MQLONG    Version;        /* Structure version number */
  MQCHAR48  ProfileName;    /* Profile name */
  MQLONG    ObjectType;     /* Object type */
  MQLONG    Authority;      /* Authority */
  PMQZED    EntityDataPtr;  /* Address of MQZED structure identifying an
                               entity */
  MQLONG    EntityType;     /* Entity type */
  MQAUTHOPT Options;        /* Options */
};
```

Fields

MQZED - Entity descriptor:

The MQZED structure is used in a number of authorization service calls to specify the entity for which authorization is to be checked.

Table 1. summarizes the fields in the structure.

Table 244. Fields in MQZED

Field	Description
StrucId	Structure identifier
Version	Version
EntityName Ptr	Entity name
EntityDomainPtr	Entity domain pointer
SecurityId	Security identifier
CorrelationPtr	Correlation pointer

Fields

StrucId
Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZED_STRUC_ID
Identifier for entity descriptor structure.

For the C programming language, the constant MQZED_STRUC_ID_ARRAY is also defined; this has the same value as MQZED_STRUC_ID, but is an array of characters instead of a string.

Version
Type: MQLONG - input

Structure version number. The value is as follows:

MQZED_VERSION_1
Version-1 entity descriptor structure.

The following constant specifies the version number of the current version:

MQZED_CURRENT_VERSION
Current version of entity descriptor structure.

EntityNamePtr
Type: PMQCHAR - input

Profile name.

Address of entity name. This is a pointer to the name of the entity whose authorization is to be checked.

EntityDomainPtr
Type: PMQCHAR - input

Address of entity domain name. This is a pointer to the name of the domain containing the definition of the entity whose authorization is to be checked.

SecurityId
Type: MQBYTE40 - input

Authority.

Security identifier. This is the security identifier whose authorization is to be checked.

CorrelationPtr
Type: MQPTR - input

Correlation pointer. This facilitates the passing of correlational data between the authenticate user function and other appropriate OAM functions.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  PMQCHAR   EntityNamePtr;    /* Address of entity name */
  PMQCHAR   EntityDomainPtr;  /* Address of entity domain name */
  MQBYTE40  SecurityId;       /* Security identifier */
  MQPTR     CorrelationPtr;   /* Address of correlation data */
```

Fields

MQZEP - Add component entry point:

A service component starts this function, during initialization, to add an entry point to the entry point vector for that service component.

Syntax

MQZEP (*Hconfig*, *Function*, *EntryPoint*, *CompCode*, *Reason*)

Parameters

Hconfig
Type: MQHCONFIG - input

Configuration handle. This handle represents the component that is being configured for this particular installable service. It must be the same as the component passed to the component configuration function by the queue manager on the component initialization call.

Function
Type: MQLONG - input

Function identifier. Valid values for this are defined for each installable service.

If MQZEP is called more than once for the same function, the last call made provides the entry point that is used.

EntryPoint
Type: PMQFUNC - input

Function entry point. This is the address of the entry point provided by the component to perform the function.

The value NULL is valid, and indicates that the function is not provided by this component. NULL is assumed for entry points that are not defined using MQZEP.

CompCode
Type: MQLONG - output

Completion code. It must be one of the following values:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason
Type: MQLONG - output

Reason code qualifying *CompCode* .

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_FUNCTION_ERROR
(2281, X'8E9') Function identifier not valid.

MQRC_HCONFIG_ERROR
(2280, X'8E8') Configuration handle not valid.

For more information about these reason codes, see API reason codes.

C invocation

```
MQZEP (Hconfig, Function, EntryPoint, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;     /* Configuration handle */
MQLONG     Function;    /* Function identifier */
PMQFUNC    EntryPoint;  /* Function entry point */
MQLONG     CompCode;    /* Completion code */
MQLONG     Reason;      /* Reason code qualifying CompCode */
```

MQZFP - Free parameters:

The MQZFP structure is used on the MQZ_FREE_USER call for the *FreeParms* parameter. This parameter specifies data related to resource to be freed.

Table 1. summarizes the fields in the structure.

Table 245. Fields in MQZFP

Field	Description
StrucId	Structure identifier
Version	Version
Reserved	Reserved field
CorrelationPtr	Correlation pointer

Fields

StrucId

Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZIC_STRUC_ID
Identifier for identity context structure. For the C programming language, the constant MQZIC_STRUC_ID_ARRAY is also defined; this has the same value as MQZIC_STRUC_ID, but is an array of characters instead of a string.

Version

Type: MQLONG - input

Structure version number. The value is as follows:

MQZFP_VERSION_1
Version-1 free parameters structure.

The following constant specifies the version number of the current version:

MQZFP_CURRENT_VERSION
Current version of free parameters structure.

Reserved

Type: MQBYTE8 - input

Reserved field. The initial value is null.

CorrelationPtr

Type: MQPTR - input

Correlation pointer. Address of correlation data relating to the resource to be freed.

C declaration

```
typedef struct tagMQZFP MQZFP;
struct tagMQZFP {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQBYTE8   Reserved;         /* Reserved field */
  MQPTR     CorrelationPtr;   /* Address of correlation data */
};
```

Fields

MQZIC - Identity context:

The MQZIC structure is used on the MQZ_AUTHENTICATE_USER call for the *IdentityContext* parameter.

The MQZIC structure contains identity context information, which identifies the user of the application that first put the message on a queue:

- The queue manager fills the *UserIdentifier* field with a name that identifies the user, the way that the queue manager can do this depends on the environment in which the application is running.
- The queue manager fills the *AccountingToken* field with a token or number that it determined from the application that put the message.
- Applications can use the *ApplIdentityData* field for any extra information that they want to include about the user (for example, an encrypted password).

Suitably authorized applications can set the identity context using the MQZ_AUTHENTICATE_USER function.

A Windows systems security identifier (SID) is stored in the *AccountingToken* field when a message is created under WebSphere MQ for Windows. The SID can be used to supplement the *UserIdentifier* field and to establish the credentials of a user.

Table 1. summarizes the fields in the structure.

Table 246. Fields in MQZIC

Field	Description
StrucId	Structure identifier
Version	Version
UserIdentifier	User identifier
AccountingToken	Accounting token
ApplIdentityData	Application identity data

Fields

StrucId

Type: MQCHAR4 - input

Structure identifier. The value is as follows:

MQZIC_STRUC_ID
Identifier for identity context structure. For the C programming language, the constant MQZIC_STRUC_ID_ARRAY is also defined; this has the same value as MQZIC_STRUC_ID, but is an array of characters instead of a string.

Version

Type: MQLONG - input

Structure version number. The value is as follows:

MQZIC_VERSION_1
Version-1 identity context structure.

The following constant specifies the version number of the current version:

MQZIC_CURRENT_VERSION
Current version of identity context structure.

UserIdentifier
Type: MQCHAR12 - input

User identifier. This is part of the identity context of the message. *UserIdentifier* specifies the user identifier of the application that originated the message. The queue manager treats this information as character data, but does not define the format of it. For more information on the *UserIdentifier* field, see "UserIdentifier (MQCHAR12)" on page 1752.

AccountingToken
Type: MQBYTE32 - input

Accounting token. This is part of the identity context of the message. *AccountingToken* allows an application to cause work done as a result of the message to be appropriately charged. The queue manager treats this information as a string of bits and does not check its content. For more information on the *AccountingToken* field, see "AccountingToken (MQBYTE32)" on page 1708.

ApplIdentityData
Type: MQCHAR32 - input

Application data relating to identity. This is part of the identity context of the message. ApplIdentityData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. For example, it could be set by applications running with suitable user authority to indicate whether the identity data is trusted. For more information on the ApplIdentityData field, see "ApplIdentityData (MQCHAR32)" on page 1710.

C declaration

```
typedef struct tagMQZED MQZED;
struct tagMQZED {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQCHAR12  UserIdentifier;   /* User identifier */
  MQBYTE32  AccountingToken;  /* Accounting token */
  MQCHAR32  ApplIdentityData; /* Application data relating to identity */
};
```

Fields

Reference material for WebSphere MQ bridge for HTTP

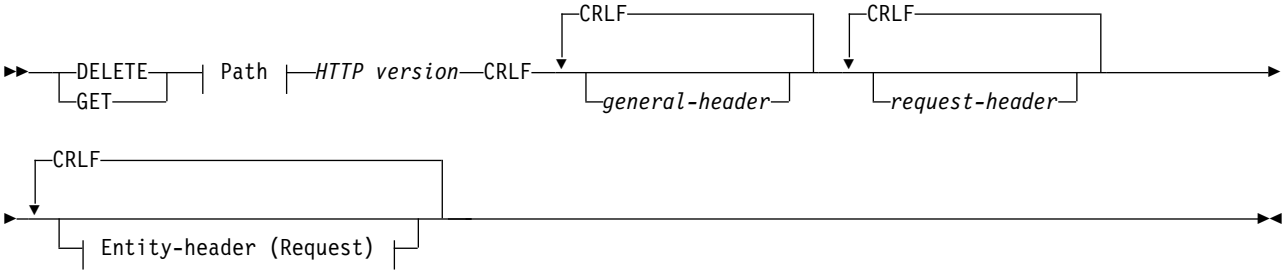
Reference topics for WebSphere MQ bridge for HTTP, arranged alphabetically

HTTP DELETE: WebSphere MQ bridge for HTTP command

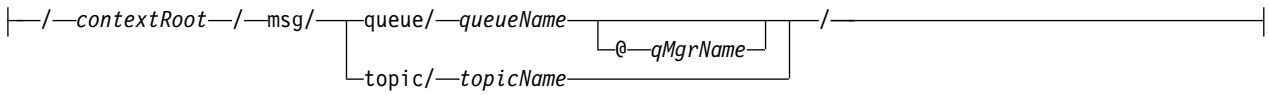
The HTTP **DELETE** operation gets a message from a WebSphere MQ queue, or retrieves a publication from a topic. The message is removed from the queue. If the publication is retained, it is not removed. A response message is sent back to the client including information about the message.

Syntax

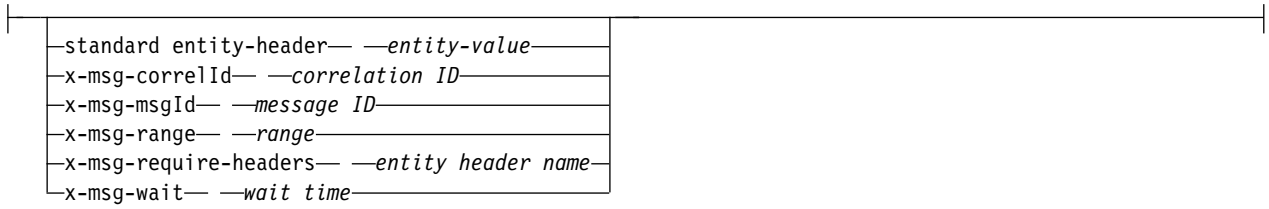
Request



Path:



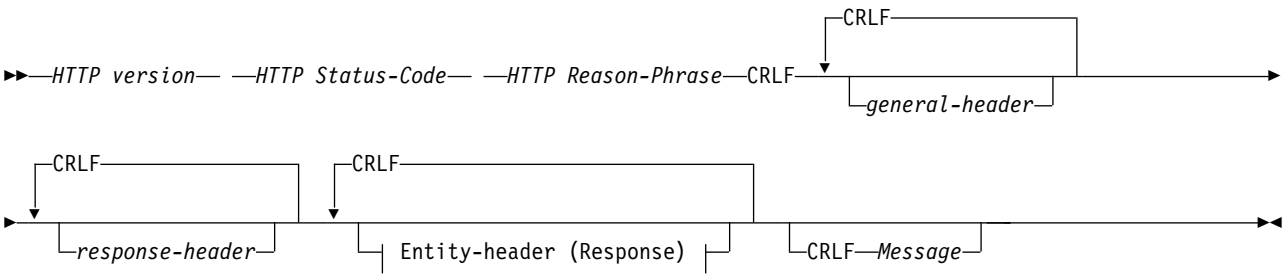
entity-header (Request):



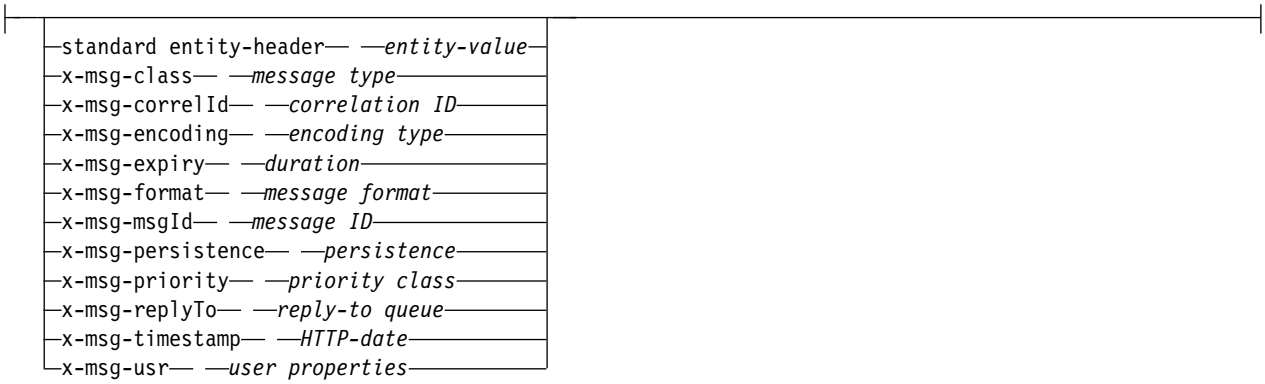
Note:

1. If a question mark (?) is used it must be substituted with `%3f`. For example, `orange?topic` should be specified as `orange%3ftopic`.
2. *@qMgrName* is only valid on an HTTP **POST**

Response



entity-header (Response):



Request parameters

Path
See "URI Format" on page 2577.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

request-header
See HTTP/1.1 - 5.3 Request Header Fields. The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

entity-header (Request)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity headers listed in the Request syntax diagram.

Response parameters

Path
See "URI Format" on page 2577.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

response-header
See HTTP/1.1 - 6.2 Response Header Fields.

entity-header (Response)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity or response headers listed in the Response syntax diagram. The `Content-Length` is always present in a response. It is set to zero if there is no message body.

Message
Message body.

Description

If the HTTP **`DELETE`** request is successful, the response message contains the data retrieved from the WebSphere MQ queue. The number of bytes in the body of the message is returned in the HTTP `Content-Length` header. The status code for the HTTP response is set to `200 OK`. If `x-msg-range` is specified as `0`, or `0-0`, then the status code of the HTTP response is `204 No Content`.

If the HTTP **DELETE** request is unsuccessful, the response includes a WebSphere MQ bridge for HTTP error message and an HTTP status code.

HTTP DELETE example

HTTP **DELETE** gets a message from a queue and deletes the message, or retrieves and deletes a publication. The **HTTPDELETE** Java sample is an example an HTTP **DELETE** request reading a message from a queue. Instead of using Java, you could create an HTTP **DELETE** request using a browser form, or an AJAX toolkit instead.

Figure 46 is an HTTP request to delete the next message on queue called `myQueue`. In response, the message body is returned to the client. In WebSphere MQ terms, HTTP **DELETE** is a destructive get.

The request contains the HTTP request header `x-msg-wait`, which instructs WebSphere MQ bridge for HTTP how long to wait for a message to arrive on the queue. The request also contains the `x-msg-require-headers` request header, which specifies that the client is to receive the message correlation ID in the response.

```
DELETE /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

Figure 46. Example of an HTTP **DELETE** *request*

Figure 47, is the response returned to the client. The correlation ID is returned to the client, as requested in `x-msg-require-headers` of the request.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that is retrieved from the queue.
```

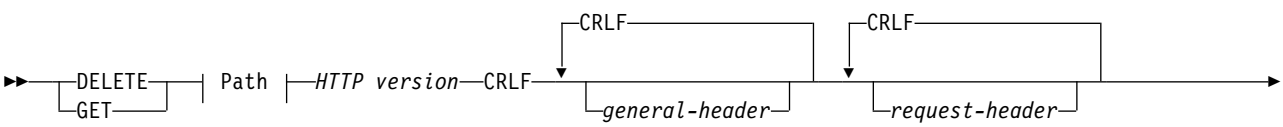
Figure 47. Example of an HTTP **DELETE** *response*

HTTP GET: WebSphere MQ bridge for HTTP command

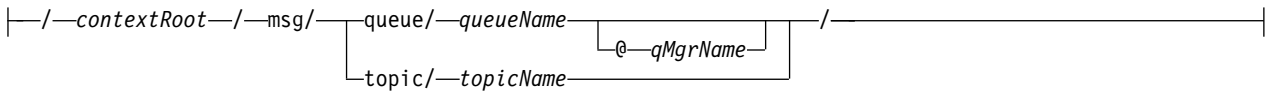
The HTTP **GET** operation gets a message from a WebSphere MQ queue. The message is left on the queue. The HTTP **GET** operation is equivalent to browsing a WebSphere MQ queue.

Syntax

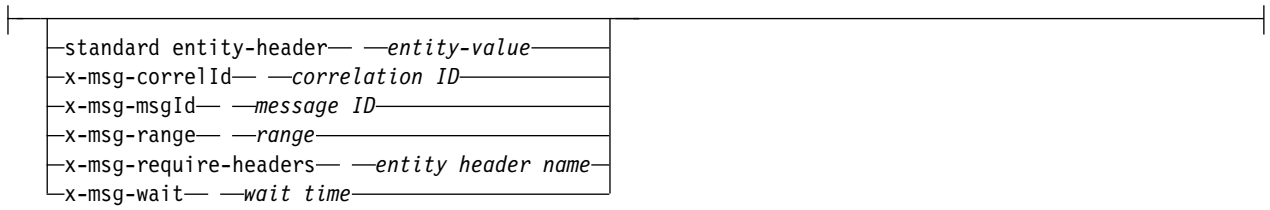
Request



Path:



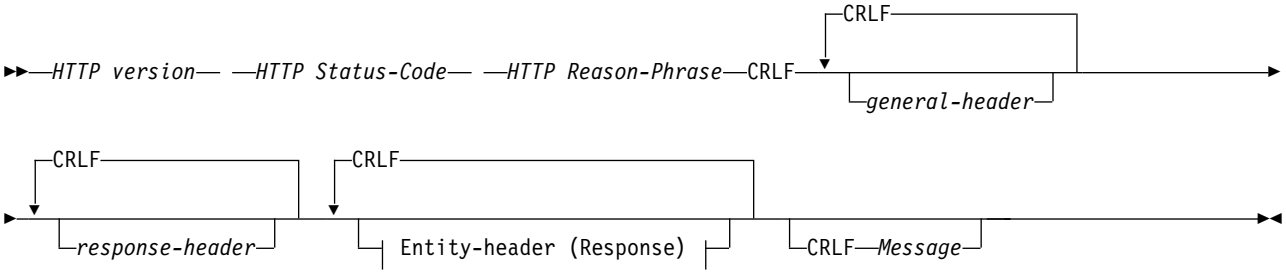
entity-header (Request):



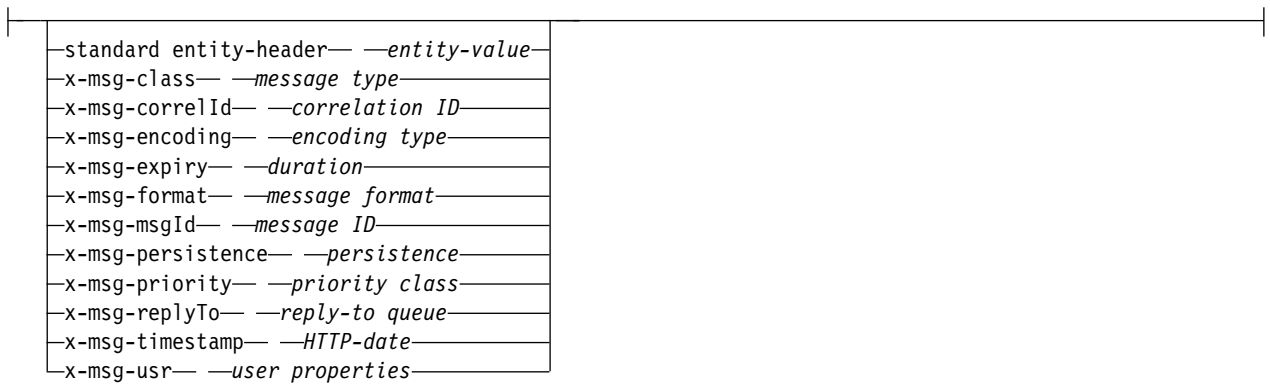
Note:

1. If a question mark (?) is used it must be substituted with `%3f`. For example, `orange?topic` should be specified as `orange%3ftopic`.
2. *@qMgrName* is only valid on an HTTP **POST**

Response



entity-header (Response):



Request parameters

Path
See "URI Format" on page 2577.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

request-header
See HTTP/1.1 - 5.3 Request Header Fields. The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

entity-header (Request)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity headers listed in the Request syntax diagram.

Response parameters

Path
See "URI Format" on page 2577.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

response-header
See HTTP/1.1 - 6.2 Response Header Fields.

entity-header (Response)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity or response headers listed in the Response syntax diagram. The `Content-Length` is always present in a response. It is set to zero if there is no message body.

Message
Message body.

Description

If the HTTP **GET** request is successful, the response message contains the data retrieved from the WebSphere MQ queue. The number of bytes in the body of the message is returned in the HTTP `Content-Length` header. The status code for the HTTP response is set to `200 OK`. If `x-msg-range` is specified as `0`, or `0-0`, then the status code of the HTTP response is `204 No Content`.

If the HTTP **GET** request is unsuccessful, the response includes a WebSphere MQ bridge for HTTP error message and an HTTP status code.

HTTP `GET` example

HTTP **GET** gets a message from a queue. The message remains on the queue. In WebSphere MQ terms, HTTP **GET** is a browse request. You could create an HTTP **GET** request using a Java client, a browser form, or an AJAX toolkit.

Figure 48 on page 2548 is an HTTP request to browse the next message on queue called `myQueue`.

The request contains the HTTP request header `x-msg-wait`, which instructs WebSphere MQ bridge for HTTP how long to wait for a message to arrive on the queue. The request also contains the

x-msg-require-headers request header, which specifies that the client is to receive the message correlation ID in the response.

```
GET /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
x-msg-wait: 10
x-msg-require-headers: correlID
```

Figure 48. Example of an HTTP **GET** *request*

Figure 49 is the response returned to the client. The correlation ID is returned to the client, as requested in x-msg-require-headers of the request.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 50
Content-Type: text/plain; charset=utf-8
x-msg-correlId: 1234567890

Here is my message body that appears on the queue.
```

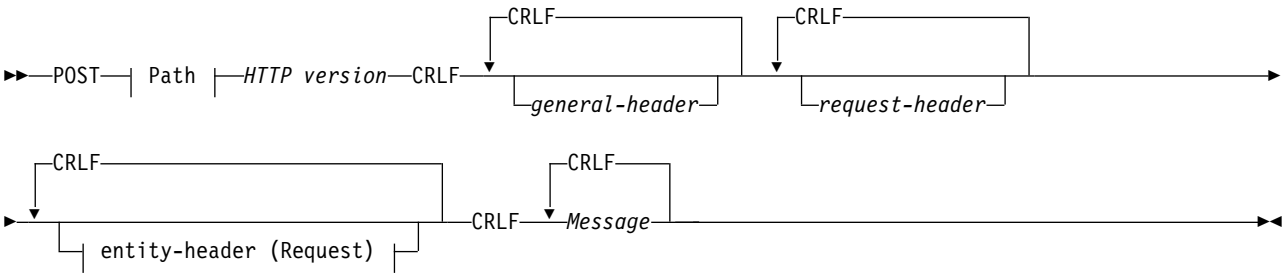
Figure 49. Example of an HTTP **GET** *response*

HTTP POST: WebSphere MQ bridge for HTTP command

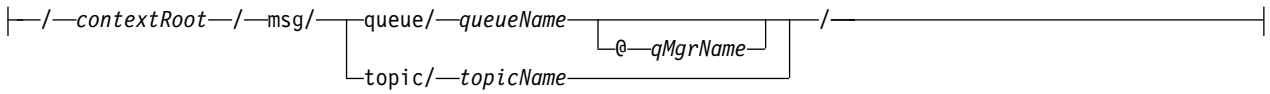
The HTTP **POST** operation puts a message on a WebSphere MQ queue, or publishes a message to a topic.

Syntax

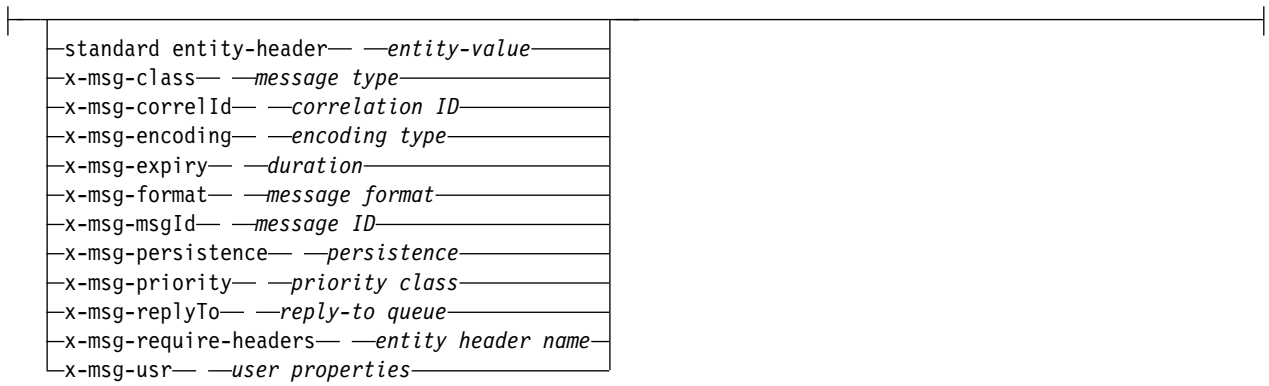
Request



Path:



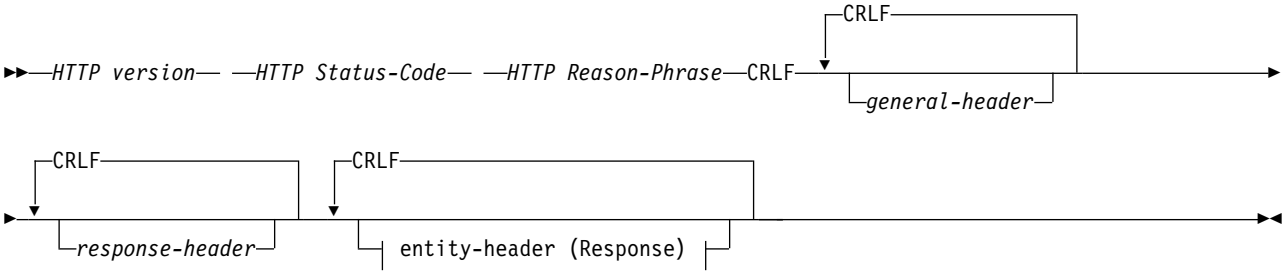
entity-header (Request):



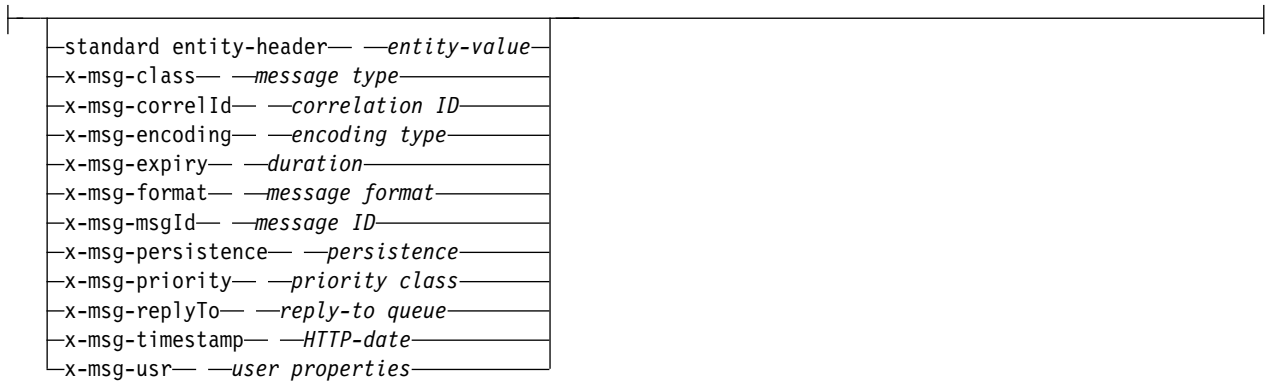
Note:

1. If a question mark (?) is used it must be substituted with `%3f`. For example, `orange?topic` should be specified as `orange%3ftopic`.
2. *@qMgrName* is only valid on an HTTP **POST**

Response



entity-header (Response):



Request parameters

Path
See "URI Format" on page 2577.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

request-header
See HTTP/1.1 - 5.3 Request Header Fields. The Host field is mandatory on an HTTP/1.1 request. It is often automatically inserted by the tool you use to create a client request.

entity-header (Request)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity headers listed in the Request syntax diagram. The `Content-Length` and `Content-Type` should be inserted in a request, and are often inserted automatically by the tool you use to create a client request. The `Content-Type` must match the type defined in the `x-msg-class` custom entity-header, if it is specified.

Message
Message to put onto the queue, or publication to post to a topic.

Response parameters

Path
See "URI Format" on page 2577.

HTTP version
HTTP version; for example, `HTTP/1.1`

general-header
See HTTP/1.1 - 4.5 General Header Fields.

response-header
See HTTP/1.1 - 6.2 Response Header Fields.

entity-header (Response)
See HTTP/1.1 - 7.1 Entity Header Fields. One of the entity or response headers listed in the Response syntax diagram. The `Content-Length` is always present in a response. It is set to zero if there is no message body.

Description

If no `x-msg-usr` header is included, and message class is `BYTES` or `TEXT`, the message put on the queue does not have an `MQRFH2`.

Use HTTP entity and request headers in the HTTP **`POST`** request to set the properties of the message that is put onto the queue. You can also use `x-msg-require-headers` to request which headers are returned in the response message.

If the HTTP **`POST`** request is successful, the entity of the response message is empty and its Content-Length is zero. The HTTP status code is `200 OK`.

If the HTTP **`POST`** request is unsuccessful, the response includes a WebSphere MQ bridge for HTTP error message and an HTTP status code. The WebSphere MQ message is not put on the queue or topic.

HTTP `POST` example

HTTP **`POST`** puts a message to a queue, or a publication to a topic. The **`HTTPPOST`** Java sample is an example an HTTP **`POST`** request of a message to a queue. Instead of using Java, you could create an HTTP **`POST`** request using a browser form, or an AJAX toolkit instead.

Figure 50 on page 2551 shows an HTTP request to put a message on a queue called `myQueue`. This request contains the HTTP header `x-msg-correlId` to set the correlation ID of the WebSphere MQ message.

```
POST /msg/queue/myQueue/ HTTP/1.1
Host: www.example.org
Content-Type: text/plain
x-msg-correlID: 1234567890
Content-Length: 50

Here is my message body that is posted on the queue.
```

Figure 50. Example of an HTTP **POST** *request to a queue*

Figure 51 shows the response sent back to the client. There is no response content.

```
HTTP/1.1 200 OK
Date: Wed, 2 Jan 2007 22:38:34 GMT
Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
Content-Length: 0
```

Figure 51. Example of an HTTP POST response

HTTP headers

The WebSphere MQ bridge for HTTP supports custom request HTTP headers, custom entity HTTP headers, and a subset of standard HTTP headers.

HTTP practice is to prefix all custom headers with `x-`, the WebSphere MQ Bridge for HTTP headers are prefixed with `x-msg-`. For example, to set the priority header use `x-msg-priority`.

Note:

- Most header values are case sensitive. For example, when using the `msgId` header, `NONE` is a keyword, whereas `none` is a `msgID`.
- Misspelled headers are ignored.

Custom entity HTTP headers

The custom entity HTTP headers contain information about WebSphere MQ messages. Using entity headers, you can set values in the message descriptor (`MQMD`), or query values in the `MQMD`. An additional entity header, `x-msg-usr`, sets and returns any user property information you want to associate with a request.

You can use entity headers in different HTTP request contexts:

DELETE You can only use the `x-msg-correlId`, or `x-msg-msgId`, or both, entity headers with a **DELETE** HTTP request. The effect of the headers is to select a particular message by `MsgId` and `CorrelId` in an MQGET, and to delete the message from its queue.

GET You can only use the `x-msg-correlId`, or `x-msg-msgId`, or both, entity headers with a **GET** HTTP request. The effect of the headers is to select a particular message by `MsgId` and `CorrelId` in an MQGET for browse.

POST You can use any entity header in a **POST** HTTP request, except `x-msg-timestamp`.

x-msg-require-headers
On any **GET**, **POST** or **DELETE** HTTP request, you can add multiple entity headers inside the `x-msg-require-headers` request header, separated by commas. The effect is to return the specified entity headers in the HTTP response message, containing the value of the associated message property.

The description of each header lists in which contexts the header is processed by WebSphere MQ bridge for HTTP. For example, in Table 247, the header is processed by WebSphere MQ bridge for HTTP in an HTTP **POST** request, or in the `x-msg-require-headers` request header in either an HTTP **POST**, **GET**, or **DELETE** request. If the header is included in a context it is not allowed in, the header is ignored. No error is reported.

You can put any standard HTTP headers into requests to be processed by the Web server, or other request handlers. Similarly, the response might contain other standard HTTP headers inserted by the Web server or other response handlers.

Table 247. Example of how the allowed contexts is documented.

Valid in HTTP request message	**POST, x-msg-require-headers**

Custom request HTTP headers

The three custom request HTTP headers, `x-msg-range`, `x-msg-require-headers`, and `x-msg-wait`, pass additional information about the HTTP request to the server. They act as request modifiers. With `x-msg-range`, you can restrict the amount of message data returned in a response. With `x-msg-require-headers`, you can request the response to contain information about the result of the request. With `x-msg-wait`, you can modify the time the client waits for an HTTP response.

Standard HTTP headers

The `Host` standard HTTP request-header must be specified in an HTTP/1.1 request.

The `Content-Length` and `Content-Type` standard HTTP entity headers can be specified in a request.

The `Content-Length`, `Content-Location`, `Content-Range`, `Content-Type`, and `Server` standard HTTP entity headers can be returned in response to a request. Specify one or more of the standard HTTP headers in the `x-msg-request-header` header in the request message.

Alphabetic list of HTTP headers

`class`: HTTP `x-msg-class` entity-header:

Set or return the message type.

Type	**Description**
HTTP header name	`x-msg-class`
HTTP header type	Entity-header
Valid in HTTP request message	POST, `x-msg-require-headers`
Allowed values	**BYTES** **MAP** **STREAM** **TEXT**
Default value	BYTES

Description

- In an HTTP **POST** request, sets the type of the message created.

- Specifying the class header on a **GET** or **DELETE** returns a `400 Bad Request` with entity body of `MQHTTP40007`.
- Specified in `x-msg-require-headers`, sets `x-msg-class` in the HTTP response message to the type of a message.
- If an invalid value is specified for this header a `MQHTTP40005` message is returned.
- If the `x-msg-class` header is not specified and the content-type of the message is `application/x-www-form-urlencoded`, the data is assumed to be a JMS map object.

`Content-Length`: HTTP entity-header:

Set or return the length, in bytes, of the body of the message.

Type	Description
HTTP header name	`Content-Length`
HTTP header type	Entity-header
Valid in HTTP request message	`x-msg-require-headers`
Allowed and returned value	*`Integer value`* Length in bytes of the message body.

Description

- The `Content-Length` is optional in an HTTP request. For a **GET** or **DELETE** the length must be zero. For **POST**, if `Content-Length` is specified and it does not match the length of the message-line, the message is either truncated, or padded with nulls to the specified length.
- The `Content-Length` is always returned in the HTTP response even when there is no content, in which case the value is zero.

`Content-Location`: HTTP entity-header:

Returns the queue or topic referenced in the request, in the standard `Content-Location` header in the HTTP response message.

Type	Description
HTTP header name	`Content-Location`
HTTP header type	Entity-header
Valid in HTTP request message	`x-msg-require-headers`
Returned value	URI in the format, *`/msg/queue/queuename`* or *`/msg/topic/topicname`*

Description

- When requested in `x-msg-require-headers`, the `Content-Location` entity-header returns the queue or topic referenced in the HTTP request.

Content-Range: HTTP entity-header:

Return the range of bytes selected from a WebSphere MQ message in the Content-Range header in an HTTP response.

Type	Description
HTTP header name	Content-Range
HTTP header type	Entity-header
Valid in HTTP request message	x-msg-require-headers
Returned value	*String* Returns the lower limit, *m* and upper limit, *n* of the returned substring, and *length* of the whole message. For example, *m-n/length*

Description

•

- The Content-Range is only returned in the HTTP response when Content-Range is specified in a **GET** or **DELETE** request that contains an x-msg-range request header.
- If x-msg-range is specified on a **GET** or **DELETE** request, the range of bytes specified in the Content-Range header are returned in the response. For example, if x-msg-range: 0-60 is used in a request for a message containing 100 bytes, the content-range header holds the string 0-60/100
- An x-msg-range request also returns the content range in the x-msg-range header in the HTTP response.

Content-Type: HTTP entity-header:

Set or return the class of the JMS message in a WebSphere MQ message according the to HTTP content-type.

Type	Description
HTTP header name	Content-Type
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, x-msg-require-headers
Allowed or returned value	*media-type* For media-types that are supported see Table 248.

Table 248. Mapping between x-msg-class and HTTP Content-Type

x-msg-class	HTTP Content-Type
BYTES	application/octet-stream application/xml
TEXT	text/*
MAP	application/x-www-form-urlencoded application/xml (optional)
STREAM	application/xml (optional)

Description

- On an HTTP **POST** request, specify either the `Content-Type` or the `x-msg-class`. If you specify both, they must be consistent or an `HTTP Bad Request` exception, `Status code 400` is returned. If you omit both, the `Content-Type` and the `x-msg-class`, a `Content-Type` of `text/*` is assumed.
- The `Content-Type` is always set in the response to an HTTP **GET** or **DELETE** that has a message body. The `Content-Type` is set according to the rules in Table 249.

Table 249. Mapping message types to `x-msg-class` and `Content-Type`

Message format	JMS Message type	x-msg-class	Content-Type
Anything except `MQFMT_STRING`	None	`BYTES`	`application/octet-stream`
`MQFMT_STRING`	None	`TEXT`	`text/plain`
`MQFMT_NONE`	`jms_bytes`	`BYTES`	`application/octet-stream`
`MQFMT_NONE`	`jms_text`	`TEXT`	`text/plain`
`MQFMT_NONE`	`jms_map`	`MAP`	`application/xml`
`MQFMT_NONE`	`jms_stream`	`STREAM`	`application/xml`

correlId: HTTP x-msg-correlId entity-header:

Set or return the correlation identifier.

Type	Description
HTTP header name	`x-msg-correlId`
HTTP header type	Entity-header
Valid in HTTP request message	**DELETE**, **GET**, **POST**, `x-msg-require-headers`

Allowed values	*String value*
	For example:
	`x-msg-correlId: mycorrelationid`
	
	Strings enclosed in quotation marks are permitted; for example:
	`x-msg-correlId: "my id"`
	
	Hex value
	A hex value prefixed with `0x:`; for example:
	`x-msg-correlId: 0x:43c1d23a`
	
	The hex value following 0x: is limited to 48 characters representing 24 bytes. Additional data is ignored.

Default value	Not applicable

Description

- On an HTTP **POST** request, sets the correlation ID of the message created.
- On an HTTP **GET** or **DELETE** request, selects the message from the queue or topic. If no message exists with the specified correlation ID, an `HTTP 504 Gateway Timeout` response is returned. `x-msg-correlId` can be used with `x-msg-msgID` to select a message from a queue or topic that matches both selectors.
- Specified in `x-msg-require-headers`, sets `x-msg-corelId` in the HTTP response message to the correlation ID of a message.
- Horizontal white space is allowed after the `0x:` prefix.

Note:

- Specifying `x-msg-correlId` without a value on an HTTP **GET** or **DELETE** request; for example, `"x-msg-correlId:"`, returns the next message on the queue or topic regardless of its correlation ID.
- If you specify a selector of 24 characters or fewer, or 0x: followed by 48 characters or fewer, WebSphere MQ bridge for HTTP uses an optimized selector for improved performance.
- A JMS message selector containing JMSCorrelationID is used when selecting messages from the queue. This selector behaves as described in Selection behavior.

encoding: HTTP x-msg-encoding entity-header:

Set or return the message encoding.

Type	Description
HTTP header name	`x-msg-encoding`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`
Allowed values	A comma-separated list of the following values: **DECIMAL_NORMAL** **DECIMAL_REVERSED** **FLOAT_IEEE_NORMAL** **FLOAT_IEEE_REVERSED** **FLOAT_S390** **INTEGER_NORMAL** **INTEGER_REVERSED** For example, `x-msg-encoding: INTEGER_NORMAL,DECIMAL_NORMAL,FLOAT_IEEE_NORMAL` **Note:** The value is case-sensitive
Default value	`DECIMAL_NORMAL, FLOAT_IEEE_NORMAL, INTEGER_NORMAL`

Description

- On an HTTP **POST** request, specifies the encoding of the message created.
- On an HTTP **GET** or **DELETE** request, the `x-msg-encoding` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-encoding` in the HTTP response message to the encoding property of a message.

expiry: HTTP x-msg-expiry entity-header:

Set or return the message expiry duration.

Type	Description
HTTP header name	`x-msg-expiry`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`

Type	Description

Allowed values	**UNLIMITED**
	For example;
	`x-msg-expiry: UNLIMITED`
	Integer value
	Milliseconds before expiry.
	For example;
	`x-msg-expiry: 10000`

Default value	UNLIMITED

Description

- When set on an HTTP **POST** request, the request message expires in the time specified.
- On an HTTP **GET** or **DELETE** request, the `x-msg-expiry` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-expiry` in the HTTP response message to the expiry time of a message.
- `UNLIMITED` specifies that the message never expires.
- The expiry of a message starts from the time the message arrives on the queue, as a result network latency is ignored.
- The maximum value is limited by WebSphere MQ to 214748364700 milliseconds. If a value greater than this is specified then the maximum possible expiry time is assumed.

`format`: HTTP `x-msg-format` entity-header:

Set or return the WebSphere MQ message format.

Type	Description
HTTP header name	`x-msg-format`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`

Allowed values	**NONE** For example,
	`x-msg-format: NONE`
	String value
	Any user-defined value of up to eight characters. For example,
	`x-msg-format: myformat`

Default value	None

Description

- When set on an HTTP **POST** request, set the request message format.
- On an HTTP **GET** or **DELETE** request, the `x-msg-format` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-format` in the HTTP response message to the format of a message.
- `NONE` is case sensitive, and indicates that the message format is blank.

- The value of `x-msg-format` is used, even if it contradicts the media-type of the HTTP request. See Table 250.

Table 250. Mapping content-type and `x-msg-class` to message format

x-msg-class	**Content-type**	**Message format on queue/topic**
BYTES	• application/octet-stream • application/xml	WebSphere MQ message: MQFMT set to MQC.MQFMT_NONE
TEXT	• text/*	WebSphere MQ message: MQFMT set to MQC.MQFMT_STRING
MAP	• application/x-www-form-urlencoded • application/xml (optional)	JMSMap
STREAM	• application/xml (optional)	JMSStream

msgId: HTTP x-msg-msgId entity-header:

Set or return the message identifier.

Type	**Description**
HTTP header name	x-msg-msgId
HTTP header type	Entity-header
Valid in HTTP request message	**DELETE**, **GET**, **POST**, x-msg-require-headers

Allowed values	*String value*
	For example,
	`x-msg-msgId: mymsgid`
	Strings enclosed in quotation marks for example, `x-msg-msgId: "my id"`
	Hex value
	A hex value prefixed with `0x:`; for example,
	`x-msg-msgId: 0x:43c1d23a`

Default value	Not applicable

Description

- On an HTTP **POST** request, sets the message ID of the message created.
- On an HTTP **GET** or **DELETE** request, selects the message from the queue or topic. If no message exists with the specified message ID, an `HTTP 504 Gateway Timeout` response is returned. `x-msg-msgId` can be used with `x-msg-correlID` to select a message from a queue or topic that matches both selectors.
- Specified in `x-msg-require-headers`, returns `x-msg-msgId` in the HTTP response to the message ID of a message.
- Horizontal white space is allowed after the `0x:`prefix.

Note: Specifying `x-msg-msgId` without a value on an HTTP **GET** or **DELETE** request; for example, `"x-msg-msgId:"`, returns the next message on the queue or topic regardless of its message ID.

A JMS message selector containing JMSMessageID is used when selecting messages from the queue. This selector behaves as described in Selection behavior.

persistence: HTTP x-msg-persistence entity-header:

Set or return the message persistence.

Type	Description
HTTP header name	`x-msg-persistence`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`

Allowed values	**NON_PERSISTENT**
	The message does not survive system failures or queue manager restarts.
	For example,
	`x-msg-persistence: NON_PERSISTENT`
	PERSISTENT
	The message survives system failures and restarts of the queue manager.
	For example,
	`x-msg-persistence: PERSISTENT`
	AS_DESTINATION
	Applies to **POST** only.
	Use the default persistence of the destination as determined by the message provider.
	Note: Case sensitive

Default value	`NON_PERSISTENT`

Description

- When set on an HTTP **POST** request, set the request message persistence.
- On an HTTP **GET** or **DELETE** request, the `x-msg-persistence` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-persistence` in the HTTP response message to the persistence of a message.

priority: HTTP x-msg-priority entity-header:

Set or return the message priority.

Type	Description
HTTP header name	`x-msg-priority`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`

Type	Description

Allowed values	**LOW** For example,
	`x-msg-priority: LOW`
	MEDIUM This priority is equal to a WebSphere MQ priority level of 4. For example,
	`x-msg-priority: MEDIUM`
	HIGH For example,
	`x-msg-priority: HIGH`
	Integer value
	A string representation of an integer in the range 0 through 9; for example,
	`x-msg-priority: 3`
	AS_DESTINATION
	Applies to **POST** only.
	Use the default priority of the destination as determined by the message provider.
	Note: Case sensitive

Default value	`MEDIUM`

Description

- When set on an HTTP **POST** request, set the request message priority.
- On an HTTP **GET** or **DELETE** request, the `x-msg-priority` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-priority` in the HTTP response message to the priority of a message.

`priority`: HTTP `x-msg-priority` entity-header:

Set or return the message priority.

Type	Description
HTTP header name	`x-msg-priority`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`

Type	Description

Allowed values	**LOW** For example,
	`x-msg-priority: LOW`
	MEDIUM This priority is equal to a WebSphere MQ priority level of 4. For example,
	`x-msg-priority: MEDIUM`
	HIGH For example,
	`x-msg-priority: HIGH`
	Integer value
	A string representation of an integer in the range 0 through 9; for example,
	`x-msg-priority: 3`
	AS_DESTINATION
	Applies to **POST** only.
	Use the default priority of the destination as determined by the message provider.
	Note: Case sensitive

Default value	`MEDIUM`

Description

- When set on an HTTP **POST** request, set the request message priority.
- On an HTTP **GET** or **DELETE** request, the `x-msg-priority` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-priority` in the HTTP response message to the priority of a message.

`replyTo`: HTTP `x-msg-replyTo` entity-header:

Set or return the message reply-to queue and queue manager name.

Type	Description
HTTP header name	`x-msg-replyTo`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`

Allowed values	*URI* A point-to-point URI; for example,
	`x-msg-replyTo: /msg/queue/myReplyQueue`
	`x-msg-replyTo: /msg/queue/myReplyQueue@myReplyQueueManager`
	Note: Case sensitive

Default value	`MEDIUM`

Description

- When set on an HTTP **POST** request, set the request message replyTo destination.
- On an HTTP **GET** or **DELETE** request, the `x-msg-replyTo` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-replyTo` in the HTTP response message to the reply-to queue and queue manager name of a message.

Note: The URI in the HTTP response can include the name of the queue manager to which the WebSphere MQ bridge for HTTP is connected.

Server: HTTP response-header:

Returns information about the server and protocol the client is connected to.

Type	Description
HTTP header name	`Server`
HTTP header type	Response-header
Valid in HTTP request message	`x-msg-require-headers`

Returned value	`WMQ-HTTP/1.1 JEE-Bridge/1.1`
	
	or
	`Server: Product-token WMQ-HTTP/1.1 JEE-Bridge/1.1`

Description

- If WebSphere MQ Bridge for HTTP is deployed to an application server, the WebSphere MQ bridge for HTTP details is appended to the server response header. For example, the WebSphere MQ bridge for HTTP deployed to WebSphere Application Server Community Edition, called Apache-Coyote, gives the response:

  ```
  Server: Apache-Coyote/1.1 WMQ-HTTP/1.1 JEE-Bridge/1.1
  ```

require-headers: HTTP x-msg-require-headers request-header:

Set which headers to return in the HTTP response message.

Type	Description
HTTP header name	`x-msg-require-headers`
HTTP header type	Request-header
Valid in HTTP request message	**POST**, **GET**, **DELETE**

Type	Description

Allowed values	A comma-separated list of the entity header names:
	ALL
	ALL-USR
	class
	content-location
	correlId
	encoding
	expiry
	format
	msgId
	NO_require-headers
	persistence
	priority
	replyTo
	server
	timestamp
	usr-*property name*
	For example,
	`x-msg-require-headers: msgId`
	or,
	`x-msg-require-headers: expiry,correlId,timestamp`
	To request a specific property:
	`x-msg-require-headers: usr-myCustomProperty`
	To request all properties:
	`x-msg-require-headers: ALL-USR, ALL`

Default value	`NO_require-headers`

Description

- The value of `x-msg-require-headers` is not case-sensitive, except in the cases of the `ALL`, `NO_require-headers`, and `ALL-USR` constants, and the *property-name* variable.

timestamp: HTTP x-msg-timestamp entity-header:

Return the message time stamp.

Type	Description
HTTP header name	`x-msg-timestamp`
HTTP header type	Entity-header
Valid in HTTP request message	`x-msg-require-headers`

Type	Description
Returned value	*HTTP-date* A date in the format; day, date month year time time-zone; for example, `Sun, 06 Nov 1994 08:49:37 GMT` Defined by `RFC 822`, and updated in `RFC 1123`.
Default value	Not applicable

Description

- On an HTTP **POST**, **GET** or **DELETE** request, the `x-msg-timestamp` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-timestamp` in the HTTP response message to the timestamp of a message.

usr: HTTP x-msg-usr entity-header:

Set or return the user properties.

Type	Description
HTTP header name	`x-msg-usr`
HTTP header type	Entity-header
Valid in HTTP request message	**POST**, `x-msg-require-headers`

Allowed values	See "Syntax"; for example,
	`x-msg-usr: myProp1;5;i1,`
	`x-msg-usr: myProp2;"My String";string`

Default value	Not applicable

Description

- When set on an HTTP **POST** request, set the request message user properties.
- On an HTTP **GET** or **DELETE** request, the `x-msg-usr` header is ignored.
- Specified in `x-msg-require-headers`, sets `x-msg-usr` in the HTTP response message to user properties of a message.
- Multiple properties can be set on a message. Specify multiple comma-separated properties in a single `x-msg-usr` header, or by using two or more separate instances of the `x-msg-usr` header.
- You can request a specific property to be returned in the response to a **GET** or **DELETE** request. Specify the name of the property in the `x-msg-require-headers` header of the request, using the prefix `usr-`. For example,

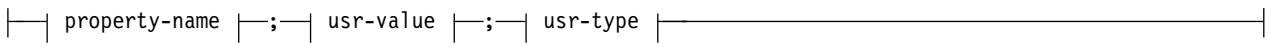
  ```
  x-msg-require-headers: usr-myProp1
  ```

- To request that all user properties are returned in a response, use the `ALL-USR` constant. For example,

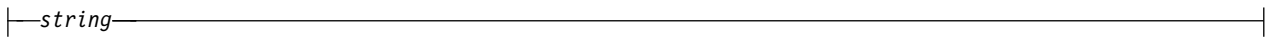
  ```
  x-msg-require-headers: ALL-USR
  ```

Syntax

usr-property-value:



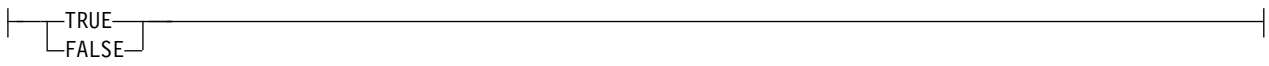
property-name:



usr-value:

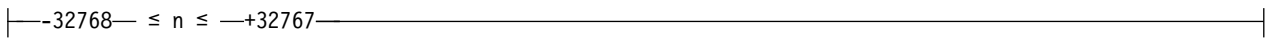
usr-type:

boolean

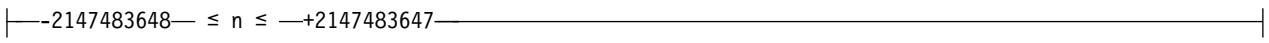


i1

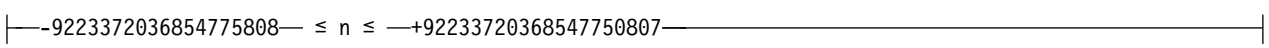
i2



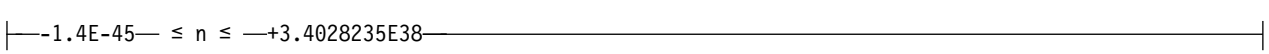
i4



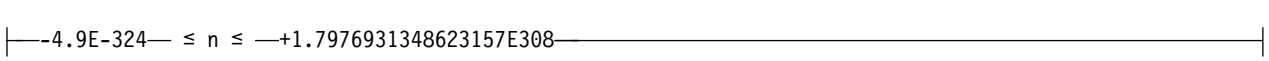
i8



r4



r8



qstring

wait: HTTP x-msg-wait request-header:

Set the period of time to wait for a message to arrive, before returning an `HTTP 504 Gateway Timeout` response message.

Type	Description
HTTP header name	`x-msg-wait`
HTTP header type	Request-header
Valid in HTTP request message	**GET**, **DELETE**

Allowed value	**NO_WAIT**
	For example,
	`x-msg-wait: NO_WAIT`
	Integer value
	The number of milliseconds that the WebSphere MQ bridge for HTTP waits for a message to arrive; for example,
	`x-msg-wait: 8`

Default value	`NO_WAIT`

Description

- On an HTTP **POST** request, the `x-msg-wait` header is ignored.
- On an HTTP **GET** or **DELETE** request, `x-msg-wait` specifies time to wait for a message to arrive before returning an `HTTP 504 Gateway Timeout` response.
- `NO_WAIT` is case-sensitive.
- The default maximum wait time is 35000. You can change the default by setting the `maximum_wait_time` parameter of the servlet. See the Installing, configuring, and verifying WebSphere MQ bridge for HTTP section for more information.

- If you set a value greater than `maximum_wait_time`, `maximum_wait_time` is used instead.

HTTP return codes

List of return codes from the WebSphere MQ bridge for HTTP

The WebSphere MQ bridge for HTTP returns four types of error:

Servlet errors
MQHTTP0001 and MQHTTP0002 are servlet errors. They are logged, but not returned to the HTTP client.

Successful operations
An HTTP status code in the range 200 - 299 indicates a successful operation.

Client errors
An HTTP status code in the range 400 - 499 indicates a client error. WebSphere MQ Bridge for HTTP return codes in the range MQHTTP40001 - MQHTTP49999 correspond to client errors.

Server errors

An HTTP status code in the range 500 - 599 indicates a client error. WebSphere MQ Bridge for HTTP return codes in the range MQHTTP50001 - MQHTTP59999 correspond to server errors.

If a server error occurs, a complete stack trace is output to the application server error log. The stack trace is also returned to the HTTP client in the HTTP response. Handle the stack trace in the client application or refer it to the application server administrator to resolve the problem.

If the stack trace contains resource adapter errors, refer to the documentation for your resource adapter.

Alphabetic list of return codes

HTTP 200: OK:

This class of status code indicates that the request was successfully received, understood and accepted.

HTTP status code

```
200 OK
```

HTTP 204: No content:

Sent following a successful HTTP **`GET`** or **`DELETE`** and `x-msg-range: 0` was sent in the request.

HTTP status code

```
204 No Content
```

MQHTTP0001: No connection factory specified in the Servlet context:

Servlet error

Explanation

Servlet error

HTTP status code

None

Programmer response

Where these errors are logged is specific to your application server. Refer to the documentation for your application server.

MQHTTP0002: Could not get connection manager for *queueOrTopic* using the JNDI name of *jndiNameTried*:

Servlet error

Explanation

Servlet error

HTTP status code

None

Programmer response

Where these errors are logged is specific to your application server. Refer to the documentation for your application server.

MQHTTP40001: Reserved:

Reserved

HTTP status code

`400 Bad Request`

MQHTTP40002: URI is not valid for WebSphere MQ transport for HTTP:

The URI specified in the HTTP request is not valid.

Explanation

The URI specified in the HTTP request is not valid.

HTTP status code

`400 Bad Request`

Programmer response

Confirm that the format and syntax of the specified URI are correct.

MQHTTP40003: URI is not valid. `@qmgr` is only valid on POST:

The `@qmgr` URI option has been specified in a URI for an HTTP request that is not a **`POST`** request.

Explanation

The `@qmgr` URI option has been specified in a URI for an HTTP request that is not a **`POST`** request.

HTTP status code

`400 Bad Request`

Programmer response

If you are attempting to put a message by using the **POST** verb, change the HTTP request to a **POST** request. If you are attempting to get a message by using the **DELETE** or **GET** verbs, remove `@qmgr` from the URI.

MQHTTP40004: Invalid `Content-Type` specified:

The `Content-Type` header field specified on a **POST** request is not compatible with the `x-msg-class` header value.

Explanation

The `Content-Type` header field specified on a **POST** request is not compatible with the `x-msg-class` header value.

HTTP status code

`400 Bad Request`

Programmer response

Change the `Content-Type` header field to one that is supported. The `Content-Type` header must be compatible with the specified `x-msg-class` header field.

MQHTTP40005: Bad message header value:

A supported header field has been specified with a value that is not valid for the specified request.

Explanation

A supported header field has been specified with a value that is not valid for the specified request.

HTTP status code

`400 Bad Request`

Programmer response

Change the value specified for the given header field to a value which is valid. Check the case of the value specified, as some header fields have case-sensitive values.

MQHTTP40006: *Header_name* is not a valid request header:

A header that is valid only in an HTTP response message has been specified in an HTTP request message.

Explanation

A header which is valid only in an HTTP response message has been specified in an HTTP request message.

HTTP status code

400 Bad Request

Programmer response

Remove any headers from the HTTP request which are only valid in an HTTP response; for example, x-msg-timestamp.

MQHTTP40007: *Header_name* is only valid on ...:

A header has been specified in an HTTP request, but the header field is not valid for the given request verb.

Explanation

A header has been specified in an HTTP request, but the header field is not valid for the given request verb.

HTTP status code

400 Bad Request

Programmer response

Remove any headers from the HTTP request which are not valid for the given request verb. For example, x-msg-encoding is valid for HTTP **POST** requests, but not valid for HTTP **GET** or HTTP **DELETE** requests.

MQHTTP40008: *Header_name* maximum length is ...:

The maximum length for the given header field has been exceeded.

Explanation

The maximum length for the given header field has been exceeded.

HTTP status code

400 Bad Request

Programmer response

Change the value of the header field to a value which is within the range permitted for the header field.

MQHTTP40009: Header field *header_field* is not valid for ...:

A header field specified in an HTTP request is not supported by the messaging provider to which the WebSphere MQ bridge for HTTP is connected.

Explanation

A header field specified in an HTTP request is not supported by the messaging provider to which the WebSphere MQ bridge for HTTP is connected. The error occurs if a messaging provider is used that cannot support all the features of the WebSphere MQ bridge for HTTP.

HTTP status code

`400 Bad Request`

Programmer response

Remove the unsupported header from the HTTP request.

MQHTTP40010: Message with `Content-Type` *content_type* could not be parsed:

The content of the HTTP request is not compatible with the `Content-Type` of the request.

Explanation

The content of the HTTP request is not compatible with the `Content-Type` of the request. A common cause is badly formed `application/x-www-form-urlencoded` or `application/xml data`.

HTTP status code

`400 Bad Request`

Programmer response

Correct the content of the HTTP request so that it is in the correct format for the `Content-Type` of the request.

MQHTTP40301: You are forbidden from accessing ...:

The WebSphere MQ bridge for HTTP has been unable to authenticate itself for the specified destination.

Explanation

The WebSphere MQ bridge for HTTP has been unable to authenticate itself for the specified destination.

HTTP status code

`403 Forbidden`

Programmer response

Change the authentication properties of the destination so that the WebSphere MQ Bridge for HTTP is authorized to connect to it. Alternatively, specify a destination to which the WebSphere MQ bridge for HTTP is authorized to connect.

MQHTTP40302: You are forbidden from ...:

The WebSphere MQ bridge for HTTP has been unable to connect to the queue manager.

Explanation

The WebSphere MQ bridge for HTTP has been unable to connect to the queue manager. The WebSphere MQ bridge for HTTP security configuration is incorrect.

HTTP status code

`403 Forbidden`

Programmer response

Change the authentication configuration of the queue manager so that the WebSphere MQ Bridge for HTTP is authorized to connect to it. Alternatively, configure the WebSphere MQ bridge for HTTP to connect to a queue manager to which it is authorized to connect.

MQHTTP40401: The destination *destination_name* could not be found:

The destination specified in the HTTP request URI cannot be found by the WebSphere MQ bridge for HTTP.

Explanation

The destination specified in the HTTP request URI cannot be found by the WebSphere MQ bridge for HTTP.

HTTP status code

```
404 Not found
```

Programmer response

Check that the destination specified in the HTTP request URI exists, or specify an alternative destination.

MQHTTP40501: Method *method_name*not allowed:

The method specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.

Explanation

The method specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.

HTTP status code

```
405 Method not allowed
```

Programmer response

Change the method specified in the HTTP request to one which is supported by the WebSphere MQ bridge for HTTP.

MQHTTP41301: The message being posted was too large for the destination:

The destination specified in the HTTP POST request URI cannot accept messages that are as long as the message specified in the HTTP request.

Explanation

The destination specified in the HTTP POST request URI cannot accept messages that are as long as the message specified in the HTTP request.

HTTP status code

```
413 Request entity too large
```

Programmer response

Reduce the size of the message specified in the HTTP request. Alternatively, specify a destination which can support messages of the wanted length.

MQHTTP41501: The media type character set is unsupported:

The character set specified in the `Content-Type` header field is not supported by the WebSphere MQ bridge for HTTP.

Explanation

The character set specified in the `Content-Type` header field is not supported by the WebSphere MQ bridge for HTTP.

HTTP status code

```
415 Unsupported media type
```

Programmer response

Change the character set of the `Content-Type` header field to one that is supported by the WebSphere MQ bridge for HTTP.

MQHTTP41502: Media-type *media-type* is not supported ...:

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified HTTP verb.

Explanation

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified HTTP verb.

HTTP status code

```
415 Unsupported media type
```

Programmer response

Change the media-type specified in the HTTP request to one that is supported by the WebSphere MQ Bridge for HTTP for the specified HTTP verb.

MQHTTP41503: Media-type *media-type* is not supported ...:

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified `x-msg-class` header field.

Explanation

The media-type specified in the HTTP request is not supported by the WebSphere MQ bridge for HTTP for the specified `x-msg-class` header field.

HTTP status code

```
415 Unsupported media type
```

Programmer response

Change the media-type specified in the HTTP request to one that is supported by the WebSphere MQ Bridge for HTTP for the specified `x-msg-class` header field.

MQHTTP41701: The HTTP header `Expect` is not supported:

The WebSphere MQ bridge for HTTP does not support the `Expect` header field.

Explanation

The `Expect` header has been specified in an HTTP request. The WebSphere MQ bridge for HTTP does not support the `Expect` header field.

HTTP status code

```
417 Expectation failed
```

Programmer response

Remove the `Expect` header from the HTTP request.

MQHTTP50001: There has been an unexpected problem ...:

An error has occurred in the WebSphere MQ bridge for HTTP.

Explanation

An error has occurred in the WebSphere MQ bridge for HTTP.

HTTP status code

```
500 Internal server error
```

Programmer response

Contact the system administrator of the WebSphere MQ Bridge for HTTP.

MQHTTP50201: An error has occurred between the WebSphere MQ bridge for HTTP and the queue manager:

An error has occurred between the WebSphere MQ bridge for HTTP and the queue manager

Explanation

An error has occurred between the WebSphere MQ bridge for HTTP and the queue manager

HTTP status code

```
502 Bad Gateway
```

Programmer response

Contact the system administrator of the WebSphere MQ Bridge for HTTP.

MQHTTP50401: Message retrieval timed out:

No message matching the specified request parameters in an HTTP **GET** or HTTP **DELETE** was returned in the timeout period.

Explanation

No message matching the specified request parameters in an HTTP **GET** or HTTP **DELETE** was returned in the timeout period. The return code indicates that no suitable message was available at any time during the life of the HTTP request.

HTTP status code

`504 Gateway timeout`

Programmer response

If a message was expected, check the header fields of the HTTP request such as `x-msg-correlId` and `x-msg-msgid`. Check that the destination specified in the HTTP request URI is correct. Try extending the wait time of the HTTP request using the `x-msg-wait` header field.

MQHTTP50501: HTTP 1.1 and upwards ...:

The HTTP protocol used in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.

Explanation

The HTTP protocol used in the HTTP request is not supported by the WebSphere MQ bridge for HTTP.

HTTP status code

`505 HTTP version not supported`

Programmer response

Change the HTTP request to use HTTP protocol V1.1 or higher.

Message types and message mappings for WebSphere Bridge for HTTP

WebSphere MQ bridge for HTTP supports four message classes, `TEXT`, `BYTES`, `STREAM` and `MAP`. The message classes are mapped to JMS message types and HTTP `Content-Type`.

HTTP POST

The message type that arrives at the destination depends on the value of the `x-msg-class` header or the `Content-Type` of the HTTP request. Table 251 on page 2576 shows the HTTP `Content-Type` type that corresponds to each `x-msg-class`. Either field can be used to set the message type and message format. If both fields are set, and are set inconsistently, then a `Bad Request exception` is returned (`HTTP 400, MQHTTP20004`).

Table 251. Mapping between x-msg-class and HTTP Content-Type

x-msg-class	HTTP Content-Type
BYTES	application/octet-stream application/xml
TEXT	text/*
MAP	application/x-www-form-urlencoded application/xml (optional)
STREAM	application/xml (optional)

If the JMS message type is set in the MQRFH2 header, it is mapped according to Table 252.

Table 252. Mapping between x-msg-class and JMS message types.

x-msg-class	JMS message type
BYTES	jms_bytes
TEXT	jms_text
MAP	jms_map
STREAM	jms_stream

The JMS message type is always set for a message class of MAP or STREAM. It is not always set for a message class of BYTES or TEXT. If a MQRFH2 is to be created for the request, the JMS message type is always set. Otherwise, if no MQRFH2 is created, no JMS message type is set. An MQRFH2 is created if user properties are set in the request, using the x-msg-usr header.

If the JMS message type is set, then the message format is set to MQFMT_NONE, see Table 254 on page 2577:

Table 253. Mapping between x-msg-class and WebSphere MQ message format

x-msg-class	Message format with MQRFH2 present in message	Message format with *no* MQRFH2 present in message
BYTES	MQFMT_NONE	MQFMT_NONE
TEXT	MQFMT_NONE	MQFMT_STRING
MAP	MQFMT_NONE	Not possible
STREAM	MQFMT_NONE	Not possible

HTTP GET or DELETE

The message type or format retrieved determines the value of the x-msg-class header and the Content-Type of the HTTP response. The x-msg-class header is returned only if requested in a x-msg-headers request.

Table 254 on page 2577 describes the mappings between x-msg-class and Content-Type, and the message type retrieved from the queue or topic.

Table 254. Mapping message types to `x-msg-class` and `Content-Type`

Message format	JMS Message type	x-msg-class	Content-Type
Anything except MQFMT_STRING	None	BYTES	application/octet-stream
MQFMT_STRING	None	TEXT	text/plain
MQFMT_NONE	jms_bytes	BYTES	application/octet-stream
MQFMT_NONE	jms_text	TEXT	text/plain
MQFMT_NONE	jms_map	MAP	application/xml
MQFMT_NONE	jms_stream	STREAM	application/xml

MAP and STREAM message class serialization

MAP and STREAM message classes are serialized back to the client in the HTTP response in the same way as a message is serialized to a queue.

For MAP, the XML name, type, and value triplets are encoded as:

```
<map>
 <elt name="elementname1" dt="datatype1">value1</elt>
 <elt name="elementname2" dt="datatype2">value2</elt>
 ...
</map>
```

STREAM is like MAP, but it does not have element names:

```
<stream>
 <elt dt="datatype1">value1</elt>
 <elt dt="datatype2">value2</elt>
 ...
</stream>
```

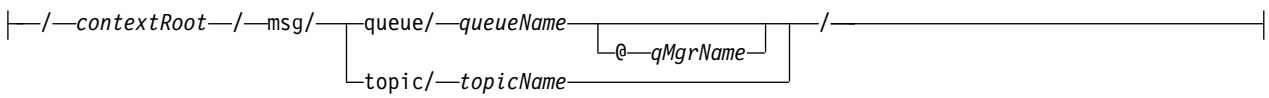
Note: datatype is one of the data types defined for defining user-defined properties and listed in "usr: HTTP x-msg-usr entity-header" on page 2564. The attribute dt="string" is omitted for string elements because the default data type is string.

URI Format

URIs intercepted by WebSphere MQ bridge for HTTP.

Syntax

Path:



Note:

1. If a question mark (?) is used it must be substituted with %3f. For example, orange?topic should be specified as orange%3ftopic.
2. *@qMgrName* is only valid on an HTTP **POST**

Description

Deploy the WebSphere MQ bridge for HTTP servlet to your JEE application server with a context root of *contextRoot*. Requests to

`http://hostname:port/context_root/msg/queue/queueName@qMgrName`

and

`http://hostname:port/context_root/msg/topic/topicString`

are intercepted by WebSphere MQ bridge for HTTP.

The WebSphere MQ .NET classes and interfaces

WebSphere MQ .NET classes and interfaces are listed alphabetically. The properties, methods and constructors are described.

MQAsyncStatus .NET class

Use MQAsyncStatus to inquire on the status of previous MQI activity; for example inquiring on the success of previous asynchronous put operations. MQAsyncStatus encapsulates features of the MQSTS data structure.

Class

```
System.Object
  |
  +- IBM.WMQ.MQBase
     |
     +- IBM.WMQ.MQBaseObject
        |
        +- IBM.WMQ.MQAsyncStatus
```

```
public class IBM.WMQ.MQAsyncStatus extends IBM.WMQ.MQBaseObject;
```

- "Properties"
- "Constructors" on page 2579

Properties

Test for MQException being thrown when getting properties.

`public static int CompCode {get;}`

The completion code from the first error or warning.

`public static int Reason {get;}`

The reason code from the first error or warning.

`public static int PutSuccessCount {get;}`

The number of successful asynchronous MQI put calls.

`public static int PutWarningCount {get;}`

The number of asynchronous MQI put calls that succeeded with a warning.

`public static int PutFailureCount {get;}`

The number of failed asynchronous MQI put calls.

`public static int ObjectType {get;}`

The object type for the first error. The following values are possible:

- `MQC.MQOT_ALIAS_Q`

- MQC.MQOT_LOCAL_Q
- MQC.MQOT_MODEL_Q
- MQC.MQOT_Q
- MQC.MQOT_REMOTE_Q
- MQC.MQOT_TOPIC
- 0, meaning that no object is returned

public static string ObjectName {get;}

The object name.

public static string ObjectQMgrName {get;}

The object queue manager name.

public static string ResolvedObjectName {get;}

The resolved object name.

public static string ResolvedObjectQMgrName {get;}

The resolved object queue manager name.

Constructors

public MQAsyncStatus() throws MQException;

Constructor method, constructs an object with fields initialized to zero or blank as appropriate.

MQAuthenticationInformationRecord .NET class

Use MQAuthenticationInformationRecord to specify information about an authenticator that is to be used in a WebSphere MQ SSL client connection. MQAuthenticationInformationRecord encapsulates an authentication information record, MQAIR.

Class

```
System.Object
  |
  └─ IBM.WMQ.MQAuthenticationInformationRecord
```

```
public class IBM.WMQ.MQAuthenticationInformationRecord extends System.Object;
```

- "Properties"
- "Constructors" on page 2580

Properties

Test for MQException being thrown when getting properties.

public long Version {get; set;}

Structure version number.

public long AuthInfoType {get; set;}

The type of authentication information. This attribute must be set to one of the following values:

- OCSP - Certificate revocation status checking is done using OCSP.
- CRLLDAP - Certificate revocation status checking is done using Certificate Revocation Lists on LDAP servers.

public string AuthInfoConnName {get; set;}

The DNS name or IP address of the host on which the LDAP server is running, with an optional port number. This keyword is required.

public string LDAPPassword {get; set;}

The password associated with the distinguished name of the user who is accessing the LDAP server. This property applies only when **AuthInfoType** is set to CRLLDAP.

public string LDAPUserName {get; set;}

The distinguished name of the user who is accessing the LDAP server. When you set this property, LDAPUserNameLength and LDAPUserNamePtr are automatically set correctly. This property applies only when AuthInfoType is set to CRLLDAP.

public string OCSPResponderURL {get; set;}

The URL at which the OCSP responder can be contacted. This property applies only when AuthInfoType is set to OCSP

This field is case sensitive. It must start with the string http:// in lowercase. The rest of the URL might be case sensitive, depending on the OCSP server implementation.

Constructors

MQAuthenticationInformationRecord();

MQDestination .NET class

Use MQDestination to access methods that are common to MQQueue and MQTopic. MQDestination is an abstract base class and cannot be instantiated.

Class

```
public class IBM.WMQ.MQDestination extends IBM.WMQ.MQManagedObject;
System.Object
  |
  └─ IBM.WMQ.MQBase
     |
     └─ IBM.WMQ.MQBaseObject
        |
        └─ IBM.WMQ.MQManagedObject
           |
           └─ IBM.WMQ.MQDestination
              |
              └─ IBM.WMQ.MQQueue
              |
              └─ IBM.WMQ.MQTopic
```

- "Properties"
- "Methods" on page 2581
- "Constructors" on page 2582

Properties

Test for MQException being thrown when getting properties.

public DateTime CreationDateTime {get;}

The date and time that the queue or topic was created. Originally contained within MQQueue, this property has been moved into the base MQDestination class.

There is no default value.

public int DestinationType {get;}

Integer value describing the type of destination being used. Initialized from the sub classes constructor, MQQueue or MQTopic, this value can take one of these values:

- MQOT_Q
- MQOT_TOPIC

There is no default value.

Methods

```
public void Get(MQMessage message);
public void Get(MQMessage message, MQGetMessageOptions getMessageOptions);
public void Get(MQMessage message, MQGetMessageOptions getMessageOptions, int MaxMsgSize);
```

Throws MQException.

Gets a message from a queue if the destination is an MQQueue object or from a topic if the destination is an MQTopic object, using a default instance of MQGetMessageOptions to do the get.

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor and message data portions of the MQMessage are replaced with the message descriptor and message data from the incoming message.

All calls to WebSphere MQ from a particular MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further WebSphere MQ calls until the Get call is accomplished. If you need multiple threads to access WebSphere MQ simultaneously, each thread must create its own MQQueueManager object.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

getMessageOptions
Options controlling the action of the get.

Using option MQC.MQGMO_CONVERT might result in an exception with reason code MQC.MQRC_CONVERTED_STRING_TOO_BIG when converting from single-byte character codes to double byte codes. In this case, the message is copied into the buffer without conversion.

If *getMessageOptions* is not specified, the message option used is MQGMO_NOWAIT.

If you use the MQGMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

MaxMsgSize
The largest message this message object is to receive. If the message on the queue is larger than this size, one of two things occurs:

- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is set in the MQGetMessageOptions object, the message is filled with as much of the message data as possible. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_ACCEPTED reason code.
- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is not set, the message is left on the queue. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_FAILED reason code.

If *MaxMsgSize* is not specified, the whole message is retrieved.

```
public void Put(MQMessage message);
public void Put(MQMessage message, MQPutMessageOptions putMessageOptions);
```

Throws MQException.

Puts a message to a queue if the destination is an MQQueue object or publishes a message to a topic if the destination is an MQTopic object.

Modifications to the MQMessage object after the Put call has been accomplished do not affect the actual message on the WebSphere MQ queue or publication topic.

Put updates the `MessageId` and `CorrelationId` properties of the MQMessage object and does not clear message data. Further Put or Get calls refer to the updated information in the MQMessage object. For example, in the following code snippet, the first message contains `a` and the second `ab`.

```
msg.WriteString("a");
q.Put(msg,pmo);
msg.WriteString("b");
q.Put(msg,pmo);
```

message
An MQMessage object containing the message descriptor data, and message to be sent. The message descriptor can be altered as a consequence of this method. The values in the message descriptor immediately after the completion of this method are the values that were put to the queue or published to the topic.

The following reason codes are returned to a reconnectable client:

- `MQRC_CALL_INTERRUPTED` if the connection is broken while running a Put call on a persistent message and the reconnection is successful.
- `MQRC_NONE` if the connection is successful while running a Put call on a non-persistent message (see Application Recovery).

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the `MQPMO_LOGICAL_ORDER` option in a reconnectable client, the `MQRC_RECONNECT_INCOMPATIBLE` reason code is returned.

Note: For simplicity and performance, if you want to put a single message to a queue, use MQQueueManager.Put object. You should have an MQQueue object for this.

Constructors

MQDestination is an abstract base class and cannot be instantiated. Access destinations using MQQueue and MQTopic constructors, or using MQQueueManager.AccessQueue and MQQueueManager.AccessTopic methods.

MQEnvironment .NET class

Use MQEnvironment to control how the MQQueueManager constructor is called and to select a WebSphere MQ MQI client connection. The MQEnvironment class contains properties that control the behaviour of the WebSphere MQ.

Class

```
System.Object
  |
  └─ IBM.WMQ.MQEnvironment
```

```
public class IBM.WMQ.MQEnvironment extends System.Object;
```

- "Properties - client only"
- "Properties" on page 2584
- "Constructors" on page 2585

Properties - client only

Test for MQException being thrown when getting properties.

public static int CertificateValPolicy {get; set;}

Set which SSL/TLS certificate validation policy is used to validate digital certificates received from remote partner systems. Valid values are:

- MQC.CERTIFICATE_VALIDATION_POLICY_ANY
- MQC.CERTIFICATE_VALIDATION_POLICY_RFC5280

public static ArrayList EncryptionPolicySuiteB {get; set;}

Set the level of Suite B compliant cryptography. Valid values are:

- MQC.MQ_SUITE_B_NONE - This is the default value.
- MQC.MQ_SUITE_B_128_BIT
- MQC.MQ_SUITE_B_192_BIT

public static string Channel {get; set;}
The name of the channel to connect to the target queue manager. You *must* set the channel property before instantiating an MQQueueManager instance in client mode.

public static int FipsRequired {get; set;}

Specify `MQC.MQSSL_FIPS_YES` to use only FIPS-certified algorithms if cryptography is carried out in WebSphere MQ. The default is `MQC.MQSSL_FIPS_NO`.

If cryptographic hardware is configured, the cryptographic modules used are those provided by the hardware product. Depending on the hardware in use, these might not be FIPS-certified to a particular level.

public static string Hostname {get; set;}
The TCP/IP host name of the computer on which the WebSphere MQ server resides. If the host name is not set, and no overriding properties are set, server bindings mode is used to connect to the local queue manager.

public static int Port {get; set;}
The port to connect to. This is the port on which the WebSphere MQ server is listening for incoming connection requests. The default value is 1414.

public static string SSLCipherSpec {get; set;}
Set `SSLCipherSpec` to the value of the CipherSpec set on the SVRCONN channel to enable SSL for the connection. The default is Null, and SSL is not enabled for the connection.

public static string sslPeerName {get; set;}

A distinguished name pattern. If `sslCipherSpec` is set, this variable can be used to ensure that the correct queue manager is used. If set to null (default), the DN of the queue manager is not performed. `sslPeerName` is ignored if `sslCipherSpec` is null.

Properties

Test for MQException being thrown when getting properties.

public static ArrayList HdrCompList {get; set;}
Header Data Compression List

public static int KeyResetCount {get; set;}
Indicates the number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated.

public static ArrayList MQAIRArray {get; set;}
An array of MQAuthenticationInformationRecord objects.

public static ArrayList MsgCompList {get; set;}
Message Data Compression List

public static string Password {get; set;}
The password to be authenticated. The password referenced from the MQCSP structure gets populated by setting this Password property.

public static string ReceiveExit {get; set;}
A receive exit allows you to examine and alter data received from a queue manager. It is normally used with a corresponding send exit at the queue manager. If ReceiveExit is set to null, no receive exit is called.

public static string ReceiveUserData {get; set;}
The user data associated with a receive exit. Limited to 32 characters.

public static string SecurityExit {get; set;}
A security exit allows you to customize the security flows that occur when an attempt is made to connect to a queue manager. If SecurityExit is set to null, no security exit is called.

public static string SecurityUserData {get; set;}
The user data associated with a security exit. Limited to 32 characters.

public static string SendExit {get; set;}
A send exit allows you to examine or alter the data sent to a queue manager. It is normally used with a corresponding receive exit at the queue manager. If SendExit is set to null, no send exit is called.

public static string SendUserData {get; set;}
The user data associated with a send exit. Limited to 32 characters.

public static string SharingConversations {get; set;}

The SharingConversations field is used on connections from .NET applications, when these applications are not using a client channel definition table (CCDT).

SharingConversations determines the maximum number of conversations that can be shared on a socket associated with this connection.

A value of 0 means that the channel operates as it did before WebSphere MQ Version 7.0, with regard to conversation sharing, read ahead, and heartbeat.

The field is passed in the hash table of properties as a SHARING_CONVERSATIONS_PROPERTY, when instantiating a WebSphere MQ queue manager.

If you do not specify SharingConversations, a default value of 10 is used.

public static string SSLCryptoHardware {get; set;}

Sets the name of the parameter string required to configure the cryptographic hardware present on the system. SSLCryptoHardware is ignored if sslCipherSpec is null.

public static string SSLKeyRepository {get; set;}

Set the fully qualified file name of the key repository.

If SSLKeyRepository is set to null (default), the certificate MQSSLKEYR environment variable is used to locate the key repository. SSLCryptoHardware is ignored if sslCipherSpec is null.

Note: The .kdb extension is a mandatory part of the file name, but is not included as part of the value of the parameter. The directory you specify must exist. WebSphere MQ creates the file the first time it accesses the new key repository, unless the file already exists.

public static string UserId {get; set;}

The user ID to be authenticated. The user ID referenced from the MQCSP structure gets populated by setting UserId. Authenticate UserId using an API or Security exit.

Constructors

public MQEnvironment()

MQException .NET class

Use MQException to find out the completion and reason code of a failed WebSphere MQ function. An MQException is thrown whenever a WebSphere MQ error occurs.

Class

```
System.Object
  |
  └─ System.Exception
     |
     └─ System.ApplicationException
        |
        └─ IBM.WMQ.MQException
```

```
public class IBM.WMQ.MQException extends System.ApplicationException;
```

- "Properties"
- "Constructors"

Properties

public int CompletionCode {get; set;}

The WebSphere MQ completion code associated with the error. The possible values are:

- MQException.MQCC_OK
- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

public int ReasonCode {get; set;}

WebSphere MQ reason code describing the error.

Constructors

public MQException(int *completionCode*, **int** *reasonCode*)

completionCode
The WebSphere MQ completion code.

reasonCode
The WebSphere MQ completion code.

MQGetMessageOptions .NET class

Use MQGetMessageOptions to specify how messages are retrieved. It modifies the behavior of MQDestination.Get.

Class

```
System.Object
  |
  +- IBM.WMQ.MQBase
     |
     +- IBM.WMQ.MQBaseObject
        |
        +- IBM.WMQ.MQGetMessageOptions
```

```
public class IBM.WMQ.MQGetMessageOptions extends IBM.WMQ.MQBaseObject;
```


- "Properties"
- "Constructors" on page 2589

Properties

Note: The behavior of some of the options available in this class depends on the environment in which they are used. These elements are marked with an asterisk *.

Test for MQException being thrown when getting properties.

public int GroupStatus {get;}*
: GroupStatus indicates whether the retrieved message is in a group and if it is the last in the group. Possible values are:

  MQC.MQGS_LAST_MSG_IN_GROUP
  : Message is the last or only message in the group.

  MQC.MQGS_MSG_IN_GROUP
  : Message is in a group, but is not the last in the group.

  MQC.MQGS_NOT_IN_GROUP
  : Message is not in a group.

public int MatchOptions {get; set;}*
: MatchOptions determines how a message is selected. The following match options can be set:

  MQC.MQMO_MATCH_CORREL_ID
  : Correlation ID to be matched.

  MQC.MQMO_MATCH_GROUP_ID
  : Group ID to be matched.

  MQC.MQMO_MATCH_MSG_ID
  : Message ID to be matched.

  MQC.MQMO_MATCH_MSG_SEQ_NUMBER
  : Match message sequence number.

  MQC.MQMO_NONE
  : No matching required.

public int Options {get; set;}
: Options control the action of MQQueue.get. Any of the following values can be specified. If more than one option is required, the values can be added, or combined using the bitwise OR operator.

  MQC.MQGMO_ACCEPT_TRUNCATED_MSG
  : Allow truncation of message data.

MQC.MQGMO_ALL_MSGS_AVAILABLE*
Retrieve messages from a group only when all the messages in the group are available.

MQC.MQGMO_ALL_SEGMENTS_AVAILABLE*
Retrieve the segments of a logical message only when all the segments in the group are available.

MQC.MQGMO_BROWSE_FIRST
Browse from start of queue.

MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR*
Browse message under browse cursor.

MQC.MQGMO_BROWSE_NEXT
Browse from the current position in the queue.

MQC.MQGMO_COMPLETE_MSG*
Retrieve only complete logical messages.

MQC.MQGMO_CONVERT
Request the application data to be converted, to conform to the `CharacterSet` and `Encoding` attributes of the MQMessage, before the data is copied into the message buffer. Because data conversion is also applied when the data is retrieved from the message buffer, applications do not set this option.

Using this option can cause problems when converting from single-byte character sets to double-byte character sets. Instead, do the conversion using the readString, readLine, and writeString methods after the message has been delivered.

MQC.MQGMO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQGMO_LOCK*
Lock the message that is browsed.

MQC.MQGMO_LOGICAL_ORDER*
Return messages in groups, and segments of logical messages, in logical order.

If you use the `MQGMO_LOGICAL_ORDER` option in a reconnectable client, the `MQRC_RECONNECT_INCOMPATIBLE` reason code is returned to the application.

MQC.MQGMO_MARK_SKIP_BACKOUT*
Allow a unit of work to be backed out without reinstating the message on the queue.

MQC.MQGMO_MSG_UNDER_CURSOR
Get message under browse cursor.

MQC.MQGMO_NONE
No other options have been specified; all options assume their default values.

MQC.MQGMO_NO_PROPERTIES
No properties of the message, except properties contained in the message descriptor (or extension) are retrieved.

MQC.MQGMO_NO_SYNCPOINT
Get message without sync point control.

MQC.MQGMO_NO_WAIT
Return immediately if there is no suitable message.

MQC.MQGMO_PROPERTIES_AS_Q_DEF
Retrieve message properties as defined by the `PropertyControl` attribute of MQQueue. Access to the message properties in the message descriptor, or extension, are not affected by the `PropertyControl` attribute.

MQC.MQGMO_PROPERTIES_COMPATIBILITY
Retrieve message properties with a prefix of mcd, jms, usr, or mqext, in MQRFH2 headers. Other properties of the message, except properties contained in the message descriptor, or extension, are discarded.

MQC.MQGMO_PROPERTIES_FORCE_MQRFH2
Retrieve message properties, except properties contained in the message descriptor, or extension, in MQRFH2 headers. Use MQC.MQGMO_PROPERTIES_FORCE_MQRFH2 in applications that are expecting to retrieve properties but cannot be changed to use message handles.

MQC.MQGMO_PROPERTIES_IN_HANDLE
Retrieve message properties using a MsgHandle.

MQC.MQGMO_SYNCPOINT
Get the message under sync point control. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

MQC.MQGMO_SYNCPOINT_IF_PERSISTENT*
Get message with sync point control if message is persistent.

MQC.MQGMO_UNLOCK*
Unlock a previously locked message.

MQC.MQGMO_WAIT
Wait for a message to arrive.

public string ResolvedQueueName {get;}
The queue manager sets ResolvedQueueName to the local name of the queue from which the message was retrieved. ResolvedQueueName is different from the name used to open the queue if an alias queue or model queue was opened.

public char Segmentation {get;}*
Segmentation indicates whether you can allow segmentation for the retrieved message. Possible values are:

MQC.MQSEG_INHIBITED
Do not allow segmentation.

MQC.MQSEG_ALLOWED
Allow segmentation

public byte SegmentStatus {get;}*
SegmentStatus is an output field that indicates whether the retrieved message is a segment of a logical message. If the message is a segment, the flag indicates whether it is the last segment. Possible values are:

MQC.MQSS_LAST_SEGMENT
Message is the last or only segment of the logical message.

MQC.MQSS_NOT_A_SEGMENT
Message is not a segment.

MQC.MQSS_SEGMENT
Message is a segment, but is not the last segment of the logical message.

public int WaitInterval {get; set;}
WaitInterval is the maximum time in milliseconds that an MQQueue.get call waits for a suitable message to arrive. Use WaitInterval with MQC.MQGMO_WAIT. Set a value of MQC.MQWI_UNLIMITED to wait an unlimited time for a message.

Constructors

public MQGetMessageOptions()

Construct a new MQGetMessageOptions object with `Options` set to `MQC.MQGMO_NO_WAIT`, `WaitInterval` set to zero, and `ResolvedQueueName` set to blank.

MQManagedObject .NET class

Use MQManagedObject to inquire and set attributes of MQDestination, MQProcess, MQQueueManager, and MQSubscription. MQManagedObject is a superclass of these classes.

Classes

```
System.Object
  |
  └─ IBM.WMQ.MQBase
    |
    └─ IBM.WMQ.MQBaseObject
      |
      └─ IBM.WMQ.MQManagedObject
        |
        └─ IBM.WMQ.MQProcess
        |
        └─ IBM.WMQ.MQQueueManager
        |
        └─ IBM.WMQ.MQSubscription
        |
        └─ IBM.WMQ.MQDestination
```

```
public class IBM.WMQ.MQManagedObject extends IBM.WMQ.MQBaseObject;
```

- "Properties"
- "Methods" on page 2590
- "Constructors" on page 2591

Properties

Test for MQException being thrown when getting properties.

public string AlternateUserId {get; set;}

The alternate user ID, if any, set when the resource was opened. `AlternateUserID.set` is ignored when issued for an object that is opened. `AlternateUserId` is not valid for subscriptions.

public int CloseOptions {get; set;}

Set this attribute to control the way the resource is closed. The default value is `MQC.MQCO_NONE`. `MQC.MQCO_NONE` s the only permissible value for all resources other than permanent dynamic queues, temporary dynamic queues, subscriptions, and topics that are being accessed by the objects that created them.

For queues and topics, the following additional values are permissible:

MQC.MQCO_DELETE

Delete the queue if there are no messages.

MQC.MQCO_DELETE_PURGE

Delete the queue, purging any messages on it.

MQC.MQCO_QUIESCE

Request the queue be closed, receiving a warning if any messages remain (allowing them to be retrieved before final closing).

For subscriptions, the following additional values are permissible:

MQC.MQCO_KEEP_SUB
The subscription is not deleted. This option is valid only if the original subscription is durable. MQC.MQCO_KEEP_SUB is the default value for a durable topic.

MQC.MQCO_REMOVE_SUB
The subscription is deleted. MQC.MQCO_REMOVE_SUB is the default value for a non-durable, unmanaged topic.

MQC.MQCO_PURGE_SUB
The subscription is deleted. MQC.MQCO_PURGE_SUB is the default value for a non-durable, managed topic.

public MQQueueManager ConnectionReference {get;}
The queue manager to which this resource belongs.

public string MQDescription {get;}
The description of the resource as held by the queue manager. MQDescription returns an empty string for subscriptions and topics.

public boolean IsOpen {get;}
Indicates whether the resource is currently open.

public string Name {get;}
The name of the resource. The name is either the supplied on the access method, or the allocated by the queue manager for a dynamic queue.

public int OpenOptions {get; set;}
OpenOptions are set when an WebSphere MQ object is opened. The OpenOptions.set method is ignored and does not cause an error. Subscriptions have no OpenOptions.

Methods

public virtual void Close();
Throws MQException.

Closes the object. No further operations against this resource are permitted after calling Close. To change the behavior of the Close method, set the closeOptions attribute.

public string GetAttributeString(int *selector*, **int** *length*);
Throws MQException.

Gets an attribute string.

selector
Integer indicating which attribute is being queried.

length
Integer indicating the length of the string required.

public void Inquire(int[] *selectors*, **int[]** *intAttrs*, **byte[]** *charAttrs*);
Throws MQException.

Returns an array of integers and a set of character strings containing the attributes of a queue, process, or queue manager. The attributes to be queried are specified in the selectors array.

Note: Many of the more common attributes can be queried using the Get methods defined in MQManagedObject, MQQueue and MQQueueManager.

selectors
Integer array identifying the attributes with values to be inquired on.

intAttrs
The array in which the integer attribute values are returned. Integer attribute values are returned in the same order as the integer attribute selectors in the selectors array.

charAttrs
The buffer in which the character attributes are returned, concatenated. Character attributes are returned in the same order as the character attribute selectors in the selectors array. The length of each attribute string is fixed for each attribute.

public void Set(int[] *selectors*, **int[]** *intAttrs*, **byte[]** *charAttrs*);
Throws MQException.

Sets the attributes defined in the vector of selectors. The attributes to be set are specified in the selectors array.

selectors
Integer array identifying the attributes with values to be set.

intAttrs
The array of integer attribute values to be set. These values must be in the same order as the integer attribute selectors in the selectors array.

charAttrs
The buffer in which the character attributes to be set are concatenated. These values must be in the same order as the character attribute selectors in the selectors array. The length of each character attribute is fixed.

public void SetAttributeString(int *selector*, **string** *value*, **int** *length*);
Throws MQException.

Sets an attribute string.

selector
Integer indicating which attribute is being set.

value
The string to set as the attribute value.

length
Integer indicating the length of the string required.

Constructors

protected MQManagedObject()
Constructor method. This object is an abstract base class which cannot be instantiated by itself.

MQMessage .NET class

Use MQMessage to access the message descriptor and data for a WebSphere MQ message. MQMessage encapsulates a WebSphere MQ message.

Class

```
System.Object
  |
  +- IBM.WMQ.MQBase
     |
     +- IBM.WMQ.MQBaseObject
        |
        +- IBM.WMQ.MQMessage
```

```
public class IBM.WMQ.MQMessage extends IBM.WMQ.MQBaseObject;
```

Create an MQMessage object and then use the Read and Write methods to transfer data between the message and other objects in your application. Send and receive MQMessage objects using the Put and Get methods of the MQDestination, MQQueue and MQTopic classes.

Get and set the properties of the message descriptor using the properties of MQMessage. Set and Get extended message properties using the SetProperty and GetProperty methods.

Properties

Test for MQException being thrown when getting properties.

public string AccountingToken {get; set;}
Part of the identity context of the message; it helps an application to charge for work done as a result of the message. The default value is `MQC.MQACT_NONE`.

public string ApplicationIdData {get; set;}
Part of the identity context of the message. `ApplicationIdData` is information that is defined by the application suite, and can be used to provide additional information about the message or its originator. The default value is `""`.

public string ApplicationOriginData {get; set;}
Information defined by the application that can be used to provide additional information about the origin of the message. The default value is `""`.

public int BackoutCount {get;}
A count of the number of times the message has previously been returned and backed out by an MQQueue.Get call as part of a unit of work. The default value is zero.

public int CharacterSet {get; set;}

The coded character set identifier of character data in the message.

Set `CharacterSet` to identify the character set of character data in the message. Get `CharacterSet` to find out in what character set has been used to encode character data in the message.

.NET applications always run in Unicode, whereas in other environments applications run in the same character set as the queue manager is running under.

The ReadString and ReadLine methods convert the character data in the message to Unicode for you.

The WriteString method converts from Unicode to the character set encoded in `CharacterSet`. If `CharacterSet` is set to its default value, `MQC.MQCCSI_Q_MGR`, which is 0, no conversion takes place and `CharacterSet` is set to 1200. If you set `CharacterSet` to some other value, WriteString converts from Unicode to the alternate value.

Note: Other read and write methods do not use `CharacterSet`.

- ReadChar and WriteChar read and write a Unicode character to and from the message buffer without conversion.
- ReadUTF and WriteUTF convert between a Unicode string in the application, and a UTF-8 string, prefixed by a 2-byte length field, in the message buffer.
- Byte methods transfer bytes between the application and the message buffer without alteration.

public byte[] CorrelationId {get; set;}

- For an MQQueue.Get call, the correlation identifier of the message to be retrieved. The queue manager returns the first message with a message identifier and a correlation identifier that match the message descriptor fields. The default value, `MQC.MQCI_NONE`, helps any correlation identifier to match.

- For an MQQueue.Put call, the correlation identifier to set.

public int DataLength {get;}
The number of bytes of message data remaining to be read.

public int DataOffset {get; set;}
The current cursor position within the message data. Reads and writes take effect at the current position.

public int Encoding {get; set;}
The representation used for numeric values in the application message data. `Encoding` applies to binary, packed decimal, and floating point data. The behavior of the read and write methods for these numeric formats is altered accordingly. Construct a value for the encoding field by adding one value from each of these three sections. Alternatively, construct the value combining the values from each of the three sections using the bitwise OR operator.

1. Binary integer

   MQC.MQENC_INTEGER_NORMAL
   Big-endian integers.

   MQC.MQENC_INTEGER_REVERSED
   Little-endian integers, as used in Intel architecture.

2. Packed-decimal

   MQC.MQENC_DECIMAL_NORMAL
   Big-endian packed-decimal, as used by z/OS.

   MQC.MQENC_DECIMAL_REVERSED
   Little-endian packed-decimal.

3. Floating-point

   MQC.MQENC_FLOAT_IEEE_NORMAL
   Big-endian IEEE floats.

   MQC.MQENC_FLOAT_IEEE_REVERSED
   Little-endian IEEE floats, as used Intel architecture.

   MQC.MQENC_FLOAT_S390
   z/OS format floating points.

The default value is:

```
MQC.MQENC_INTEGER_REVERSED |
MQC.MQENC_DECIMAL_REVERSED |
MQC.MQENC_FLOAT_IEEE_REVERSED
```

The default setting causes WriteInt to write a little-endian integer, and ReadInt to read a little-endian integer. If you set the flag `MQC.MQENC_INTEGER_NORMAL` flag instead, WriteInt writes a big-endian integer, and ReadInt reads a big-endian integer.

Note: A loss in precision can occur when converting from IEEE format floating points to zSeries format floating points.

public int Expiry {get; set;}
An expiry time expressed in tenths of a second, set by the application that puts the message. After the expiry time of a message has elapsed, it is eligible to be discarded by the queue manager. If the message specified one of the `MQC.MQRO_EXPIRATION` flags, a report is generated when the message is discarded. The default value is `MQC.MQEI_UNLIMITED`, meaning that the message never expires.

public int Feedback {get; set;}
Use `Feedback` with a message of type `MQC.MQMT_REPORT` to indicate the nature of the report. The following feedback codes are defined by the system:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQC.MQFB_PAN
- MQC.MQFB_NAN
- MQC.MQFB_DATA_LENGTH_ZERO
- MQC.MQFB_DATA_LENGTH_NEGATIVE
- MQC.MQFB_DATA_LENGTH_TOO_BIG
- MQC.MQFB_BUFFER_OVERFLOW
- MQC.MQFB_LENGTH_OFF_BY_ONE
- MQC.MQFB_IIH_ERROR

Application-defined feedback values in the range MQC.MQFB_APPL_FIRST to MQC.MQFB_APPL_LAST can also be used. The default value of this field is MQC.MQFB_NONE, indicating that no feedback is provided.

public string Format {get; set;}

A format name used by the sender of the message to indicate the nature of the data in the message to the receiver. You can use your own format names, but names beginning with the letters MQ have meanings that are defined by the queue manager. The queue manager built-in formats are:

MQC.MQFMT_ADMIN
Command server request/reply message.

MQC.MQFMT_COMMAND_1
Type 1 command reply message.

MQC.MQFMT_COMMAND_2
Type 2 command reply message.

MQC.MQFMT_DEAD_LETTER_HEADER
Dead-letter header.

MQC.MQFMT_EVENT
Event message.

MQC.MQFMT_NONE
No format name.

MQC.MQFMT_PCF
User-defined message in programmable command format.

MQC.MQFMT_STRING
Message consisting entirely of characters.

MQC.MQFMT_TRIGGER
Trigger message

MQC.MQFMT_XMIT_Q_HEADER
Transmission queue header.

The default value is MQC.MQFMT_NONE.

public byte[] GroupId {get; set;}

A byte string that identifies the message group to which the physical message belongs. The default value is MQC.MQGI_NONE.

public int MessageFlags {get; set;}
Flags controlling the segmentation and status of a message.

public byte[] MessageId {get; set;}

For an MQQueue.Get call, this field specifies the message identifier of the message to be retrieved. Normally, the queue manager returns the first message with a message identifier and correlation identifier that match the message descriptor fields. Allow any message identifier to match using the special value MQC.MQMI_NONE.

For an MQQueue.Put call, this field specifies the message identifier to use. If MQC.MQMI_NONE s specified, the queue manager generates a unique message identifier when the message is put. The value of this member variable is updated after the put, to indicate the message identifier that was used. The default value is MQC.MQMI_NONE.

public int MessageLength {get;}
The number of bytes of message data in the MQMessage object.

public int MessageSequenceNumber {get; set;}
The sequence number of a logical message within a group.

public int MessageType {get; set;}
Indicates the type of the message. The following values are currently defined by the system:

- MQC.MQMT_DATAGRAM
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT
- MQC.MQMT_REQUEST

Application-defined values can also be used, in the range MQC.MQMT_APPL_FIRST to MQC.MQMT_APPL_LAST. The default value of this field is MQC.MQMT_DATAGRAM.

public int Offset {get; set;}
In a segmented message, the offset of data in a physical message from the start of a logical message.

public int OriginalLength {get; set;}
The original length of a segmented message.

public int Persistence {get; set;}

Message persistence. The following values are defined:

- MQC.MQPER_NOT_PERSISTENT

  If you set this option in a reconnectable client, theMQRC_NONE reason code is returned to the application when the connection is successful.
- MQC.MQPER_PERSISTENT

  If you set this option in a reconnectable client, the MQRC_CALL_INTERRUPTED reason code is returned to the application after the connection is successful.
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

The default value is MQC.MQPER_PERSISTENCE_AS_Q_DEF, which takes the persistence for the message from the default persistence attribute of the destination queue.

public int Priority {get; set;}
The message priority. The special value MQC.MQPRI_PRIORITY_AS_Q_DEF can also be set in outbound message. The priority for the message is then taken from the default priority attribute of the destination queue. The default value is MQC.MQPRI_PRIORITY_AS_Q_DEF.

public int PropertyValidation {get; set;}
Specifies whether validation of properties takes place when a property of the message is set. Possible values are:

- MQCMHO_DEFAULT_VALIDATION
- MQCMHO_VALIDATE
- MQCMHO_NO_VALIDATION

The default value is MQCMHO_DEFAULT_VALIDATION.

public string PutApplicationName {get; set;}
The name of the application that put the message. The default value is "".

public int PutApplicationType {get; set;}
The type of application that put the message. PutApplicationType can be a system-defined or user-defined value. The following values are defined by the system:

- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS
- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA

The default value is MQC.MQAT_NO_CONTEXT, which indicates that no context information is present in the message.

public DateTime PutDateTime {get; set;}
The time and date that the message was put.

public string ReplyToQueueManagerName {get; set;}
The name of the queue manager to send reply or report messages. The default value is "", and the queue manager provides the ReplyToQueueManagerName.

public string ReplyToQueueName {get; set;}
The name of the message queue to which the application that issued the get request for the message sends MQC.MQMT_REPLY and MQC.MQMT_REPORT messages. The default ReplyToQueueName is "".

public int Report {get; set;}
Use Report to specify options about report and reply messages:

- Whether reports are required.
- Whether the application message data is to be included in the reports.
- How to set the message and correlation identifiers in the report or reply.

Any combination of the four report types can be requested:

- Specify any combination of the four report types. Selecting any of the three options for each report type, depending on whether the application message data is to be included in the report message.
  1. Confirm on arrival
     - MQC.MQRO_COA
     - MQC.MQRO_COA_WITH_DATA
     - MQC.MQRO_COA_WITH_FULL_DATA**
  2. Confirm on delivery
     - MQC.MQRO_COD

- MQC.MQRO_COD_WITH_DATA
- MQC.MQRO_COD_WITH_FULL_DATA**

3. Exception
   - MQC.MQRO_EXCEPTION
   - MQC.MQRO_EXCEPTION_WITH_DATA
   - MQC.MQRO_EXCEPTION_WITH_FULL_DATA**
4. Expiration
   - MQC.MQRO_EXPIRATION
   - MQC.MQRO_EXPIRATION_WITH_DATA
   - MQC.MQRO_EXPIRATION_WITH_FULL_DATA**

Note: Values marked with ** in the list are not supported by z/OS queue managers. Do not use them if your application is likely to access a z/OS queue manager, regardless of the platform on which the application is running.

- Specify one of the following to control how the message ID is generated for the report or reply message:
  - MQC.MQRO_NEW_MSG_ID
  - MQC.MQRO_PASS_MSG_ID
- Specify one of the following to control how the correlation ID of the report or reply message is to be set:
  - MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
  - MQC.MQRO_PASS_CORREL_ID
- Specify one of the following to control the disposition of the original message when it cannot be delivered to the destination queue:
  - MQC.MQRO_DEAD_LETTER_Q
  - MQC.MQRO_DISCARD_MSG**
- If no report options are specified, the default is:

  ```
  MQC.MQRO_NEW_MSG_ID |
  MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID |
  MQC.MQRO_DEAD_LETTER_Q
  ```

- You can specify one or both of the following to request that the receiving application sends a positive action or negative action report message.
  - MQC.MQRO_PAN
  - MQC.MQRO_NAN

public int TotalMessageLength {get;}
The total number of bytes in the message as stored on the message queue from which this message was received.

public string UserId {get; set;}
UserId is part of the identity context of the message. The queue manager generally provides the value. You can override the value if you have authority to set the identity context.

public int Version {get; set;}
The version of the MQMD structure in use.

Read and Write message methods

The Read and Write methods perform the same functions as the members of the BinaryReader and BinaryWriter classes in the .NET System.IO namespace. See MSDN for the full language syntax and usage examples. The methods read or write from the current position in the message buffer. They move the current position forward by the number of bytes read or written.

Note: If the message data contains an MQRFH or MQRFH2 header, you must use the ReadBytes method to read the data.

- All the methods throw IOException.
- The ReadFully methods automatically resize the target `byte` or `sbyte` array to fit the message exactly. A null array is also resized.
- Read methods throw EndOfStreamException.
- WriteDecimal methods throw MQException.
- ReadString, ReadLine and WriteString methods convert between Unicode and the character set of the message; see `CharacterSet`.
- The Decimal methods read and write packed decimal numbers encoded either in big-endian, `MQC.MQENC_DECIMAL_NORMAL`, or little-endian `MQC.MQENC_DECIMAL_REVERSE` format, according to the value of `Encoding`. Decimal ranges and corresponding .NET types are as follows:

  `Decimal2/short`
  -999 to 999

  `Decimal4/int`
  -9999999 to 9999999

  `Decimal8/long`
  -999999999999999 to 999999999999999
- The Double and Float methods read and write floating values encoded in IEE big-endian and little-endian formats, `MQC.MQENC_FLOAT_IEEE_NORMAL` and `MQC.MQENC_FLOAT_IEEE_REVERSED`, or in S/390 format, `MQC.MQENC_FLOAT_S390`, according to the value of `Encoding`.
- The Int methods read and write integer values encoded in big-endian, `MQC.MQENC_INTEGER_NORMAL`, or little-endian, `MQC.MQENC_INTEGER_REVERSED`, format, according to the value of `Encoding`. The integers are all signed, except for the addition of an unsigned 2-byte integer type. The integer sizes, and .NET and WebSphere MQ types are as follows:

  2 byte `short`, `Int2`, `ushort`, `UInt2`

  4 byte `int`, `Int4`

  8 byte `long`, `Int8`
- WriteObject transfers the class of an object, the values of its non-transient and non-static fields, and the fields of its supertypes, to the message buffer.
- ReadObject creates an object from the class of the object, the signature of the class, and the values of its non-transient and non-static fields, and the fields of its supertypes.

Table 255. Read and Write message methods

Target type	Method signatures
`Boolean`	`public bool ReadBoolean();` `public void WriteBoolean(bool value);`

`Byte`	`public byte ReadByte()`
	`public byte ReadUnsignedByte()`
	`public void Write(int value)`
	`public void WriteByte(int value)`
	`public void WriteByte(byte value)`
	`public void WriteByte(sbyte value)`

Table 255. Read and Write message methods (continued)

Target type	Method signatures
Bytes	public byte[] ReadBytes(int *count*) public void ReadFully(ref byte[] *value*) public void ReadFully(ref sbyte[] *value*) public void ReadFully(ref byte[] *value*, int *offset*,int *length*) public void ReadFully(ref sbyte[] *value*, int *offset*,int *length*) public void Write(byte[] *value*) public void Write(sbyte[] *value*) public void Write(byte[] *value*, int *offset*,int *length*) public void Write(sbyte[] *value*, int *offset*,int *length*) public void WriteBytes(string *value*)
Decimal2	public void WriteDecimal2(short *value*)
Decimal4	public void WriteDecimal4(short *value*)
Decimal8	public void WriteDecimal8(short *value*)
Double	public double ReadDouble() public void WriteDouble(double *value*)
Float	public float ReadFloat() public void WriteFloat(float *value*)
Int2	public void WriteInt2(int *value*)
Int4	public int readDecimal4() public int ReadInt() public int ReadInt4() public void WriteInt(int *value*) public void WriteInt4(int *value*)
Int8	public void WriteInt8(long *value*)
Long	public long ReadDecimal8() public long ReadLong() public long ReadInt8() public void WriteLong(long *value*)
Object	public Object ReadObject() public void WriteObject(Object *object*)
Short	public short ReadShort() public short ReadDecimal2() public short ReadInt2() public void WriteShort(int *value*)
string	public string ReadString(int *length*) public void WriteString(string *string*)
Unsigned Short	public ushort ReadUnsignedShort() public ushort ReadUInt2()
Unicode	public string ReadLine() public char ReadChar() public void WriteChar(int *value*) public void WriteChars(string *string*)
UTF	public string ReadUTF() public void WriteUTF(string *string*)

Buffer methods

public void ClearMessage();

Throws IOException.

Discards any data in the message buffer and sets the data offset back to zero.

public void ResizeBuffer(int *size*)

Throws IOException.

A hint to the MQMessage object about the size of buffer that might be required for subsequent get operations. If the message currently contains message data, and the new size is less than the current size, the message data is truncated.

public void Seek(int *pos*)

Throws IOException, ArgumentOutOfRangeException, ArgumentException.

Moves the cursor to the absolute position in the message buffer given by *pos*. Subsequent reads and writes act at this position in the buffer.

public int SkipBytes(int *i*)

Throws IOException, EndOfStreamException.

Moves forward `n` bytes in the message buffer and returns `n`, the number of bytes skipped.

SkipBytes method blocks until one of the following events occurs:

- All the bytes are skipped
- The end of message buffer is detected
- An exception is thrown

Property methods

public void DeleteProperty(string *name*);

Throws MQException.

Deletes a property with the specified name from the message.

name
The name of the property to delete.

public System.Collections.IEnumerator GetPropertyNames(string *name*)

Throws MQException.

Returns an IEnumerator of all the property names matching the specified name. The percent sign `'%'` can be used at the end of the name as a wildcard character to filter the properties of the message, matching on zero, or more characters, including the period.

name
The name of the property to match on.

- All the SetProperty and GetProperty methods throw MQException

Table 256. SetProperty and GetProperty methods

Type	Method signatures

Boolean	public boolean GetBooleanProperty(string *name*);
	public boolean GetBooleanProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetBooleanProperty(string *name*, boolean *value*);
	public void SetBooleanProperty(string *name*, MQPropertyDescriptor *pd*, boolean *value*);

Byte	public sbyte GetByteProperty(string *name*);
	public sbyte GetByteProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetByteProperty(string *name*, sbyte *value*);
	public void SetByteProperty(string *name*, MQPropertyDescriptor *pd*, sbyte *value*);

Bytes	public sbyte[] GetBytesProperty(string *name*);
	public sbyte[] GetBytesProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetBytesProperty(string *name*, sbyte[] *value*);
	public void SetBytesProperty(string *name*, MQPropertyDescriptor *pd*, sbyte[] *value*);

Double	public double GetDoubleProperty(string *name*);
	public double GetDoubleProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetDoubleProperty(string *name*, double *value*);
	public void SetDoubleProperty(string *name*, MQPropertyDescriptor *pd*, double *value*);

Float	public float GetFloatProperty(string *name*);
	public float GetFloatProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetFloatProperty(string *name*, float *value*);
	public void SetFloatProperty(string *name*, MQPropertyDescriptor *pd*, float *value*);

Int2	public short GetInt2Property(string *name*);
	public short GetInt2Property(string *name*, MQPropertyDescriptor *pd*);
	public void SetInt2Property(string *name*, short *value*);
	public void SetInt2Property(string *name*, MQPropertyDescriptor *pd*, short *value*);

Int4	public int GetInt4Property(string *name*);
	public int GetInt4Property(string *name*, MQPropertyDescriptor *pd*);
	public void SetInt4Property(string *name*, int *value*);
	public void SetInt4Property(string *name*, MQPropertyDescriptor *pd*, int *value*);

Int8	public long GetInt8Property(string *name*);
	public long GetInt8Property(string *name*, MQPropertyDescriptor *pd*);
	public void SetInt8Property(string *name*, long *value*);
	public void SetInt8Property(string *name*, MQPropertyDescriptor *pd*, long *value*);

Long	public long GetLongProperty(string *name*);
	public long GetLongProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetLongProperty(string *name*, long *value*);
	public void SetLongProperty(string *name*, MQPropertyDescriptor *pd*, long *value*);

Object	public Object GetObjectProperty(string *name*);
	public Object GetObjectProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetObjectProperty(string *name*, Object *value*);
	public void SetObjectProperty(string *name*, MQPropertyDescriptor *pd*, Object *value*);

Short	public short GetShortProperty(string *name*);
	public short GetShortProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetShortProperty(string *name*, short *value*);
	public void SetShortProperty(string *name*, MQPropertyDescriptor *pd*, short *value*);

string	public string GetStringProperty(string *name*);
	public string GetStringProperty(string *name*, MQPropertyDescriptor *pd*);
	public void SetStringProperty(string *name*, string *value*);
	public void SetStringProperty(string *name*, MQPropertyDescriptor *pd*, string *value*);

Constructors

public MQMessage();

Creates an MQMessage object with default message descriptor information and an empty message buffer.

MQProcess .NET class

Use MQProcess to query the attributes of a WebSphere MQ process. Create an MQProcess object using a constructor, or an MQQueueManager AccessProcess method.

Class

```
System.Object
  |
  +- IBM.WMQ.MQBase
    |
    +- IBM.WMQ.MQBaseObject
      |
      +- IBM.WMQ.MQManagedObject
        |
        +- IBM.WMQ.MQProcess
```

```
public class IBM.WMQ.MQProcess extends IBM.WMQ.MQManagedObject;
```

- "Properties"
- "Constructors" on page 2603

Properties

Test for MQException being thrown when getting properties.

public string ApplicationId {get;}

Gets the character string that identifies the application to be started. `ApplicationId` is used by a trigger monitor application. `ApplicationId` is sent to the initiation queue as part of the trigger message.

The default value is null.

public int ApplicationType {get;}

Identifies the type of the process to be started by a trigger monitor application. Standard types are defined, but others can be used:

- `MQAT_AIX`
- `MQAT_CICS`
- `MQAT_IMS`
- `MQAT_MVS`
- `MQAT_NATIVE`
- `MQAT_OS400`
- `MQAT_UNIX`
- `MQAT_WINDOWS`
- `MQAT_JAVA`
- `MQAT_USER_FIRST`
- `MQAT_USER_LAST`

The default value is `MQAT_NATIVE`.

public string EnvironmentData {get;}

Gets information about the environment of the application that is to be started.

The default value is null.

public string UserData {get;}

Gets information the user has provided about the application to be started.

The default value is null.

Constructors

public MQProcess(MQQueueManager *queueManager*, **string** *processName*, **int** *openOptions*);
public MQProcess(MQQueueManager *qMgr*, **string** *processName*, **int** *openOptions*, **string** *queueManagerName*, **string** *alternateUserId*);

Throws MQException.

Access a WebSphere MQ process on queue manager *qMgr* to inquire on process attributes.

qMgr
Queue manager to access.

processName
The name of the process to open.

openOptions
Options that control the opening of the process. The valid options that can be added, or combined using a bitwise OR, are:

- MQC.MQOO_FAIL_IF_QUIESCING
- MQC.MQOO_INQUIRE
- MQC.MQOO_SET
- MQC.MQOO_ALTERNATE_USER_AUTHORITY

queueManagerName
The name of the queue manager on which the process is defined. You can leave a blank or null queue manager name if the queue manager is the same as the one the process is accessing.

alternateUserId

If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the *openOptions* parameter, *alternateUserId* specifies the alternative user ID used to check the authorization for the action. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be blank or null.

Default user authority is used for connection to the queue manager if MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified.

public MQProcess MQQueueManager.AccessProcess(string *processName*, **int** *openOptions*);
public MQProcess MQQueueManager.AccessProcess(string *processName*, **int** *openOptions*, **string** *queueManagerName*, **string** *alternateUserId*);

Throws MQException.

Access a WebSphere MQ process on this queue manager to inquire on process attributes.

processName
The name of the process to open.

openOptions
Options that control the opening of the process. The valid options that can be added, or combined using a bitwise OR, are:

- MQC.MQOO_FAIL_IF_QUIESCING
- MQC.MQOO_INQUIRE

- MQC.MQOO_SET
- MQC.MQOO_ALTERNATE_USER_AUTHORITY

queueManagerName
The name of the queue manager on which the process is defined. You can leave a blank or null queue manager name if the queue manager is the same as the one the process is accessing.

alternateUserId

If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the *openOptions* parameter, *alternateUserId* specifies the alternative user ID used to check the authorization for the action. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be blank or null.

Default user authority is used for connection to the queue manager if MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified.

MQPropertyDescriptor .NET class

Use MQPropertyDescriptor as a parameter to MQMessage GetProperty and SetProperty methods. MQPropertyDescriptor describes an MQMessage property.

Class

```
System.Object
  |
  └ IBM.WMQ.MQPropertyDescriptor
```

```
public class IBM.WMQ.MQPropertyDescriptor extends System.Object;
```

- "Properties"
- "Constructors" on page 2605

Properties

Test for MQException being thrown when getting properties.

public int Context {get; set;}

The message context the property belongs to. Possible values are:

MQC.MQPD_NO_CONTEXT
The property is not associated with a message context.

MQC.MQPD_USER_CONTEXT
The property is associated with the user context.

If the user is authorized, a property associated with the user context is saved when a message is retrieved. A subsequent Put method referencing the saved context, can pass the property into the new message.

public int CopyOptions {get; set;}

CopyOptions describes which type of message the property can be copied into.

When a queue manager receives a message containing a WebSphere MQ defined property that the queue manager recognizes as being incorrect, the queue manager corrects the value of the CopyOptions field.

Any combination of the following options can be specified. Combine the options by adding the values, or using bitwise OR.

MQC.MQCOPY_ALL
The property is copied into all types of subsequent messages.

MQC.MQCOPY_FORWARD
The property is copied into a message being forwarded.

MQC.MQCOPY_PUBLISH
The property is copied into the message received by a subscriber when a message is being published.

MQC.MQCOPY_REPLY
The property is copied into a reply message.

MQC.MQCOPY_REPORT
The property is copied into a report message.

MQC.MQCOPY_DEFAULT
The value indicated no other copy options have been specified. No relationship exists between the property and subsequent messages. MQC.MQCOPY_DEFAULT is always returned for message descriptor properties.

MQC.MQCOPY_NONE
The same as MQC.MQCOPY_DEFAULT

public int Options { set; }
Options defaults to CMQC.MQPD_NONE. You cannot set any other value.

public int Support { get; set; }

Set Support to specify the level of support required for WebSphere MQ-defined message properties. Support for all other properties is optional. Any or none of the following values can be specified

MQC.MQPD_SUPPORT_OPTIONAL
The property is accepted by a queue manager even if it is not supported. The property can be discarded in order for the message to flow to a queue manager that does not support message properties. This value is also assigned to properties that are not WebSphere MQ defined.

MQC.MQPD_SUPPORT_REQUIRED
Support for the property is required. If you put the message to a queue manager that does not support the WebSphere MQ-defined property, the method fails. It returns completion code MQC.MQCC_FAILED and reason code MQC.MQRC_UNSUPPORTED_PROPERTY.

MQC.MQPD_SUPPORT_REQUIRED_IF_LOCAL

Support for the property is required, if the message is destined for a local queue. If you put the message to a local queue on a queue manager that does not support the WebSphere MQ-defined property, the method fails. It returns completion code MQC.MQCC_FAILED and reason code MQC.MQRC_UNSUPPORTED_PROPERTY.

No check is made if the message is put to a remote queue manager.

Constructors

PropertyDescriptor();
Create a property descriptor.

MQPutMessageOptions .NET class

Use MQPutMessageOptions to specify how messages are sent. It modifies the behavior of MQDestination.Put.

Class

```
System.Object
  |
  +- IBM.WMQ.MQBase
     |
     +- IBM.WMQ.MQBaseObject
        |
        +- IBM.WMQ.MQPutMessageOptions
```

```
public class IBM.WMQ.MQPutMessageOptions extends IBM.WMQ.MQBaseObject;
```

• "Properties""Constructors" on page 2608

Properties

Test for MQException being thrown when getting properties.

Note: The behavior of some of the options available in this class depends on the environment in which they are used. These elements are marked with an asterisk, *.

public MQQueue ContextReference {get; set;}

If the options field includes MQC.MQPMO_PASS_IDENTITY_CONTEXT or MQC.MQPMO_PASS_ALL_CONTEXT, set this field to refer to the MQQueue from which to take the context information.

The initial value of this field is null.

public int InvalidDestCount {get;}*

Generally, used for distribution lists, InvalidDestCount indicates the number of messages that could not be sent to queues in a distribution list. The count includes queues that failed to open and also the queues that were opened successfully, but for which the put operation had failed.

.NET does not support distribution lists, but InvalidDestCount is set when opening a single queue.

public int KnownDestCount {get;} *

Generally used for distribution lists, KnownDestCount indicates the number of messages that the current call has sent successfully to queues that resolve to local queues.

.NET does not support distribution lists, but InvalidDestCount is set when opening a single queue.

public int Options {get; set;}

Options that control the action of MQDestination.put and MQQueueManager.put. Any or none of the following values can be specified. If more than one option is required, the values can be added or combined using the bitwise OR operator.

MQC.MQPMO_ASYNC_RESPONSE
This option causes the MQDestination.put call to be made asynchronously, with some response data.

MQC.MQPMO_DEFAULT_CONTEXT
Associate default context with the message.

MQC.MQPMO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQPMO_LOGICAL_ORDER*
Put logical messages and segments in message groups into their logical order.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned to the application.

MQC.MQPMO_NEW_CORREL_ID*
Generate a new correlation ID for each sent message.

MQC.MQPMO_NEW_MSG_ID*
Generate a new message ID for each sent message.

MQC.MQPMO_NONE
No options specified. Do not use with other options.

MQC.MQPMO_NO_CONTEXT
No context is to be associated with the message.

MQC.MQPMO_NO_SYNCPOINT
Put a message without sync point control. If the sync point control option is not specified, a default of no sync point is assumed.

MQC.MQPMO_PASS_ALL_CONTEXT
Pass all context from an input queue handle.

MQC.MQPMO_PASS_IDENTITY_CONTEXT
Pass identity context from an input queue handle.

MQC.MQPMO_RESPONSE_AS_Q_DEF

For an MQDestination.put call, this option takes the put response type from DEFPRESP attribute of the queue.

For an MQQueueManager.put call, this option causes the call to be made synchronously.

MQC.MQPMO_RESPONSE_AS_TOPIC_DEF
MQC.MQPMO_RESPONSE_AS_TOPIC_DEF is a synonym for MQC.MQPMO_RESPONSE_AS_Q_DEF for use with topic objects.

MQC.MQPMO_RETAIN

The publication being sent is to be retained by the queue manager. If this option is used and the publication cannot be retained, the message is not published and the call fails with MQC.MQRC_PUT_NOT_RETAINED.

Request a copy of this publication after the time it was published, by calling the MQSubscription.RequestPublicationUpdate method. The saved publication is sent to applications that create a subscription without setting the MQC.MQSO_NEW_PUBLICATIONS_ONLY option. Check the MQIsRetained message property of a publication, when it is received, to find out if it was the retained publication.

When retained publications are requested by a subscriber, the subscription used might contain a wildcard in the topic string. If there are multiple retained publications in the topic tree that match the subscription, they are all sent.

MQC.MQPMO_SET_ALL_CONTEXT
Set all context from the application.

MQC.MQPMO_SET_IDENTITY_CONTEXT
Set identity context from the application.

MQC.MQPMO_SYNC_RESPONSE
This option causes the MQDestination.put or MQQueueManager.put call to be made synchronously, with full response data.

MQC.MQPMO_SUPPRESS_REPLYTO
Any information filled into the ReplyToQueueName and ReplyToQueueManagerName fields of the publication is not passed on to subscribers. If this option is used in combination with a report option that requires a ReplyToQueueName, the call fails with MQC.MQRC_MISSING_REPLY_TO_Q.

MQC.MQPMO_SYNCPOINT
Put a message with sync point control. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

public int RecordFields {get; set;} *

Information about distribution lists. Distribution lists are not supporting in .NET.

public string ResolvedQueueManagerName {get;}

An output field set by the queue manager to the name of the queue manager that owns the queue specified by the remote queue name. ResolvedQueueManagerName might be different from the name of the queue manager from which the queue was accessed if the queue is a remote queue.

A nonblank value is returned only if the object is a single queue. If the object is a distribution list or a topic, the value returned is undefined.

public string ResolvedQueueName {get;}

An output field that is set by the queue manager to the name of the queue on which the message is placed. ResolvedQueueName might be different from the name used to open the queue if the opened queue was an alias or model queue.

A non-blank value is returned only if the object is a single queue. If the object is a distribution list or a topic, the value returned is undefined.

public int UnknownDestCount {get;} *

Generally used for distribution lists, UnknownDestCount is an output field set by the queue manager. It reports the number of messages that the current call has sent successfully to queues that resolve to remote queues.

.NET does not support distribution lists, but InvalidDestCount is set when opening a single queue.

Constructors

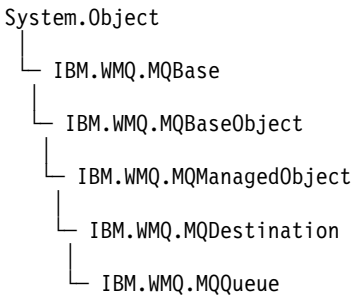
public MQPutMessageOptions();

Construct a new MQPutMessageOptions object with no options set, and a blank ResolvedQueueName and ResolvedQueueManagerName.

MQQueue .NET class

Use MQQueue to send and receive messages, and query attributes of a WebSphere MQ queue. Create an MQQueue object using a constructor, or an MQQueueManager.AccessProcess method.

Class




```
public class IBM.WMQ.MQQueue extends IBM.WMQ.MQDestination;
```


- "Properties"
- "Methods" on page 2611
- "Constructors" on page 2613

Properties

Test for MQException being thrown when getting properties.

public int ClusterWorkLoadPriority {get;}
Specifies the priority of the queue. This parameter is valid only for local, remote, and alias queues.

public int ClusterWorkLoadRank {get;}
Specifies the rank of the queue. This parameter is valid only for local, remote, and alias queues.

public int ClusterWorkLoadUseQ {get;}
Specifies the behavior of an MQPUT operation when the target queue has a local instance and at least one remote cluster instance. This parameter does not apply if the MQPUT originates from a cluster channel. This parameter is valid only for local queues.

public DateTime CreationDateTime {get;}
The date and time that this queue was created.

public int CurrentDepth {get;}
Gets the number of messages currently on the queue. This value is incremented during a put call, and during backout of a get call. It is decremented during a non-browse get and during backout of a put call.

public int DefinitionType {get;}
How the queue was defined. The possible values are:

- MQC.MQQDT_PREDEFINED
- MQC.MQQDT_PERMANENT_DYNAMIC
- MQC.MQQDT_TEMPORARY_DYNAMIC

public int InhibitGet {get; set;}
Controls whether you can get messages on this queue or for this topic. The possible values are:

- MQC.MQQA_GET_INHIBITED
- MQC.MQQA_GET_ALLOWED

public int InhibitPut {get; set;}
Controls whether you can put messages on this queue or for this topic. The possible values are:

- MQQA_PUT_INHIBITED
- MQQA_PUT_ALLOWED

public int MaximumDepth {get;}
The maximum number of messages that can exist on the queue at any one time. An attempt to put a message to a queue that already contains this many messages fails with reason code MQC.MQRC_Q_FULL.

public int MaximumMessageLength {get;}
The maximum length of the application data that can exist in each message on this queue. An attempt to put a message larger than this value fails with reason code MQC.MQRC_MSG_TOO_BIG_FOR_Q.

public int NonPersistentMessageClass {get;}
The level of reliability for non-persistent messages put to this queue.

public int OpenInputCount {get;}
The number of handles that are currently valid for removing messages from the queue. OpenInputCount is the total number of valid input handles known to the local queue manager, not just handles created by the application.

public int OpenOutputCount {get;}
The number of handles that are currently valid for adding messages to the queue. OpenOutputCount is the total number of valid output handles known to the local queue manager, not just handles created by the application.

public int QueueAccounting {get;}
Specifies whether you can enable the collection of accounting information for the queue.

public int QueueMonitoring {get;}
Specifies whether you can enable the monitoring for the queue.

public int QueueStatistics {get;}
Specifies whether you can enable the collection of statistics for the queue.

public int QueueType {get;}
The type of this queue with one of the following values:

- MQC.MQQT_ALIAS
- MQC.MQQT_LOCAL
- MQC.MQQT_REMOTE
- MQC.MQQT_CLUSTER

public int Shareability {get;}
Whether the queue can be opened for input multiple times. The possible values are:

- MQC.MQQA_SHAREABLE
- MQC.MQQA_NOT_SHAREABLE

public string TPIPE {get;}
The TPIPE name used for communication with OTMA using the WebSphere MQ IMS bridge.

public int TriggerControl {get; set;}
Whether trigger messages are written to an initiation queue, to start an application to service the queue. The possible values are:

- MQC.MQTC_OFF
- MQC.MQTC_ON

public string TriggerData {get; set;}
The free-format data that the queue manager inserts into the trigger message. It inserts TriggerData when a message arriving on this queue causes a trigger message to be written to the initiation queue. The maximum permissible length of the string is given by MQC.MQ_TRIGGER_DATA_LENGTH.

public int TriggerDepth {get; set;}
The number of messages that must be on the queue before a trigger message is written when trigger type is set to MQC.MQTT_DEPTH.

public int TriggerMessagePriority {get; set;}
The message priority under which messages do not contribute to the generation of trigger messages. That is, the queue manager ignores these messages when deciding whether to generate a trigger. A value of zero causes all messages to contribute to the generation of trigger messages.

public int TriggerType {get; set;}
The conditions under which trigger messages are written as a result of messages arriving on this queue. The possible values are:

- MQC.MQTT_NONE
- MQC.MQTT_FIRST
- MQC.MQTT_EVERY
- MQC.MQTT_DEPTH

Methods

```
public void Get(MQMessage message);
public void Get(MQMessage message, MQGetMessageOptions getMessageOptions);
public void Get(MQMessage message, MQGetMessageOptions getMessageOptions, int MaxMsgSize);
```

Throws MQException.

Gets a message from a queue.

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor and message data portions of the MQMessage are replaced with the message descriptor and message data from the incoming message.

All calls to WebSphere MQ from a particular MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further WebSphere MQ calls until the Get call is accomplished. If you need multiple threads to access WebSphere MQ simultaneously, each thread must create its own MQQueueManager object.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

getMessageOptions
Options controlling the action of the get.

Using option MQC.MQGMO_CONVERT might result in an exception with reason code MQC.MQRC_CONVERTED_STRING_TOO_BIG when converting from single-byte character codes to double byte codes. In this case, the message is copied into the buffer without conversion.

If *getMessageOptions* is not specified, the message option used is MQGMO_NOWAIT.

If you use the MQGMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

MaxMsgSize
The largest message this message object is to receive. If the message on the queue is larger than this size, one of two things occurs:

- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is set in the MQGetMessageOptions object, the message is filled with as much of the message data as possible. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_ACCEPTED reason code.
- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is not set, the message is left on the queue. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_FAILED reason code.

If *MaxMsgSize* is not specified, the whole message is retrieved.

```
public void Put(MQMessage message);
public void Put(MQMessage message, MQPutMessageOptions putMessageOptions);
```

Throws MQException.

Puts a message to a queue.

Modifications to the MQMessage object after the Put call has been accomplished do not affect the actual message on the WebSphere MQ queue or publication topic.

Put updates the MessageId and CorrelationId properties of the MQMessage object and does not clear message data. Further Put or Get calls refer to the updated information in the MQMessage object. For example, in the following code snippet, the first message contains a and the second ab.

```
msg.WriteString("a");
q.Put(msg,pmo);
msg.WriteString("b");
q.Put(msg,pmo);
```

message
An MQMessage object containing the message descriptor data, and message to be sent. The message descriptor can be altered as a consequence of this method. The values in the message descriptor immediately after the completion of this method are the values that were put to the queue or published to the topic.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if the connection is broken while running a Put call on a persistent message and the reconnection is successful.
- MQRC_NONE if the connection is successful while running a Put call on a non-persistent message (see Application Recovery).

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

Note: For simplicity and performance, if you want to put a single message to a queue, use MQQueueManager.Put object. You should have an MQQueue object for this.

```
public void PutForwardMessage(MQMessage message);
public void PutForwardMessage(MQMessage message, MQPutMessageOptions putMessageOptions);
```

Throws MQException

Put a message being forwarded onto the queue, where *message* is the original message.

message
An MQMessage object containing the message descriptor data, and message to be sent. The message descriptor can be altered as a consequence of this method. The values in the message descriptor immediately after the completion of this method are the values that were put to the queue or published to the topic.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if the connection is broken while running a Put call on a persistent message and the reconnection is successful.
- MQRC_NONE if the connection is successful while running a Put call on a non-persistent message (see Application Recovery).

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

```
public void PutReplyMessage(MQMessage message)
public void PutReplyMessage(MQMessage message, MQPutMessageOptions putMessageOptions)
```

Throws MQException.

Put a reply message onto the queue, where *message* is the original message.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

```
public void PutReportMessage(MQMessage message)
public void PutReportMessage(MQMessage message, MQPutMessageOptions putMessageOptions)
```

Throws MQException.

Put a report message onto the queue, where *message* is the original message.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

Constructors

```
public MQQueue MQQueueManager.AccessQueue(string queueName, int openOptions);
public MQQueue MQQueueManager.AccessQueue(string queueName, int openOptions, string
queueManagerName, string dynamicQueueName, string alternateUserId);
```

Throws MQException.

Accesses a queue on this queue manager.

You can get or browse messages, put messages, inquire about the attributes of the queue or set the attributes of the queue. If the queue named is a model queue, a dynamic local queue is created. Query the `name` attribute of the resultant MQQueue object to find out the name of the dynamic queue.

queueName
Name of queue to open.

openOptions
Options that control the opening of the queue.

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Validate with the specified user identifier.

MQC.MQOO_BIND_AS_QDEF
Use default binding for queue.

MQC.MQOO_BIND_NOT_FIXED
Do not bind to a specific destination.

MQC.MQOO_BIND_ON_OPEN
Bind handle to destination when queue is opened.

MQC.MQOO_BROWSE
Open to browse message.

MQC.MQOO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQOO_INPUT_AS_Q_DEF
Open to get messages using queue-defined default.

MQC.MQOO_INPUT_SHARED
Open to get messages with shared access.

MQC.MQOO_INPUT_EXCLUSIVE
Open to get messages with exclusive access.

MQC.MQOO_INQUIRE
Open for inquiry - required if you want to query properties.

MQC.MQOO_OUTPUT
Open to put messages.

MQC.MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQC.MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved.

MQC.MQOO_SET
Open to set attributes - required if you want to set properties.

MQC.MQOO_SET_ALL_CONTEXT
Allows all context to be set.

MQC.MQOO_SET_IDENTITY_CONTEXT
Allows identity context to be set.

queueManagerName
Name of the queue manager on which the queue is defined. A name that is entirely blank or null denotes the queue manager to which the MQQueueManager object is connected.

dynamicQueueName
dynamicQueueName is ignored unless queueName specifies the name of a model queue. If it does, *dynamicQueueName* specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last nonblank character in the name is an asterisk, *, the queue manager replaces the asterisk with a string of characters. The characters guarantee that the name generated for the queue is unique on this queue manager.

alternateUserId
If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, *alternateUserId* specifies the alternate user identifier that is used to check the authorization for the open. If MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be left blank, or null.

public MQQueue(MQQueueManager *queueManager*, **string** *queueName*, **int** *openOptions*, **string** *queueManagerName*, **string** *dynamicQueueName*, **string** *alternateUserId*);

Throws MQException.

Accesses a queue on queueManager.

You can get or browse messages, put messages, inquire about the attributes of the queue or set the attributes of the queue. If the queue named is a model queue, a dynamic local queue is created. Query the name attribute of the resultant MQQueue object to find out the name of the dynamic queue.

queueManager
Queue manager to access queue on.

queueName
Name of queue to open.

openOptions
Options that control the opening of the queue.

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Validate with the specified user identifier.

MQC.MQOO_BIND_AS_QDEF
Use default binding for queue.

MQC.MQOO_BIND_NOT_FIXED
Do not bind to a specific destination.

MQC.MQOO_BIND_ON_OPEN
Bind handle to destination when queue is opened.

MQC.MQOO_BROWSE
Open to browse message.

MQC.MQOO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQOO_INPUT_AS_Q_DEF
Open to get messages using queue-defined default.

MQC.MQOO_INPUT_SHARED
Open to get messages with shared access.

MQC.MQOO_INPUT_EXCLUSIVE
Open to get messages with exclusive access.

MQC.MQOO_INQUIRE
Open for inquiry - required if you want to query properties.

MQC.MQOO_OUTPUT
Open to put messages.

MQC.MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQC.MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved.

MQC.MQOO_SET
Open to set attributes - required if you want to set properties.

MQC.MQOO_SET_ALL_CONTEXT
Allows all context to be set.

MQC.MQOO_SET_IDENTITY_CONTEXT
Allows identity context to be set.

queueManagerName
Name of the queue manager on which the queue is defined. A name that is entirely blank or null denotes the queue manager to which the MQQueueManager object is connected.

dynamicQueueName
dynamicQueueName is ignored unless queueName specifies the name of a model queue. If it does, *dynamicQueueName* specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last nonblank character in the name is an asterisk, *, the queue manager replaces the asterisk with a string of characters. The characters guarantee that the name generated for the queue is unique on this queue manager.

alternateUserId
If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, *alternateUserId* specifies the alternate user identifier that is used to check the authorization for the open. If MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be left blank, or null.

MQQueueManager .NET class

Use MQQueueManager to connect to a queue manager and access queue manager objects. It also controls transactions. The MQQueueManager constructor creates either a client or server connection.

Class

```
System.Object
  |
  +- IBM.WMQ.MQBase
    |
    +- IBM.WMQ.MQBaseObject
      |
      +- IBM.WMQ.ManagedObject
        |
        +- IBM.WMQ.MQQueueManager
```

```
public class IBM.WMQ.MQQueueManager extends IBM.WMQ.MQManagedObject;
```

- "Properties" on page 2617
- "Methods" on page 2620
- " Constructors" on page 2626

Properties

Test for MQException being thrown when getting properties.

public int AccountingConnOverride {get;}
Whether applications can override the setting of the MQI accounting and queue accounting values.

public int AccountingInterval {get;}
How long before intermediate accounting records are written (in seconds).

public int ActivityRecording {get;}
Controls the generation of activity reports.

public int AdoptNewMCACheck {get;}
Specifies which elements are checked to determine whether the MCA is adopted when a new inbound channel is detected. To be adopted, the MCA name must match the name of an active MCA.

public int AdoptNewMCAInterval {get;}
The amount of time, in seconds, that the new channel waits for the orphaned channel to end.

public int AdoptNewMCAType {get;}
Whether an orphaned MCA instance is to be adopted (restarted) when a new inbound channel request is detected matching the AdoptNewMCACheck value.

public int BridgeEvent {get;}
Whether IMS Bridge events are generated.

public int ChannelEvent {get;}
Whether channel events are generated.

public int ChannelInitiatorControl {get;}
Whether the channel initiator starts automatically when the queue manager starts.

public int ChannelInitiatorAdapters {get;}
The number of adapter subtasks to process WebSphere MQ calls.

public int ChannelInitiatorDispatchers {get;}
The number of dispatchers to use for the channel initiator.

public int ChannelInitiatorTraceAutoStart {get;}
Specifies whether the channel initiator trace starts automatically.

public int ChannelInitiatorTraceTableSize {get;}
The size, in megabytes, of the trace data space of a channel initiator.

public int ChannelMonitoring {get;}
Whether channel monitoring is used.

public int ChannelStatistics {get;}
Controls the collection of statistics data for channels.

public int CharacterSet {get;}
Returns the coded character set identifier (CCSID) of the queue manager. CharacterSet is used by the queue manager for all character string fields in the application programming interface.

public int ClusterSenderMonitoring {get;}
Controls the collection of online monitoring data for automatically defined cluster sender channels.

public int ClusterSenderStatistics {get;}
Controls the collection of statistics data for automatically defined cluster sender channels.

public int ClusterWorkLoadMRU {get;}
The maximum number of outbound cluster channels.

public int ClusterWorkLoadUseQ {get;}
The default value of the MQQueue property, `ClusterWorkLoadUseQ`, if it specifies a value of `QMGR`.

public int CommandEvent {get;}
Specifies whether command events are generated.

public string CommandInputQueueName {get;}
Returns the name of the command input queue defined on the queue manager. Applications can send commands to this queue, if authorized to do so.

public int CommandLevel {get;}
Indicates the function level of the queue manager. The set of functions that correspond to a particular function level depends on the platform. On a particular platform, you can rely on every queue manager supporting the functions at the lowest functional level common to all the queue managers.

public int CommandLevel {get;}
Whether the command server starts automatically when the queue manager starts.

public string DNSGroup {get;}
The name of the group that the TCP listener handling inbound transmissions for the queue-sharing group must join. It joins this group when using Workload Manager for Dynamic Domain Name Services support (DDNS).

public int DNSWLM {get;}
Whether the TCP listener that handles inbound transmissions for the queue-sharing group must register with Workload Manager for DDNS.

public int IPAddressVersion {get;}
Which IP protocol (IPv4 or IPv6) to use for a channel connection.

public boolean IsConnected {get;}
Returns the value of the `isConnected`.

If true, a connection to the queue manager has been made, and is not known to be broken. Any calls to IsConnected do not actively attempt to reach the queue manager, so it is possible that physical connectivity can break, but IsConnected can still return true. The IsConnected state is only updated when activity, for example, putting a message, getting a message, is performed on the queue manager.

If false, a connection to the queue manager has not been made, or has been broken, or has been disconnected.

public int KeepAlive {get;}
Specifies whether the TCP KEEPALIVE facility is to be used to check that the other end of the connection is still available. If it is unavailable, the channel is closed.

public int ListenerTimer {get;}
The time interval, in seconds, between attempts by WebSphere MQ to restart the listener after an APPC or TCP/IP failure.

public int LoggerEvent {get;}
Whether logger events are generated.

public string LU62ARMSuffix {get;}
The suffix of the APPCPM member of SYS1.PARMLIB. This suffix nominates the LUADD for this channel initiator. When automatic restart manager (ARM) restarts the channel initiator, the z/OS command SET APPC=xx is issued.

public string LUGroupName {get; z/os}
The generic LU name to be used by the LU 6.2 listener that handles inbound transmissions for the queue-sharing group.

public string LUName {get;}
The name of the LU to use for outbound LU 6.2 transmissions.

public int MaximumActiveChannels {get;}
The maximum number of channels that can be active at any time.

public int MaximumCurrentChannels {get;}
The maximum number of channels that can be current at any time (including server-connection channels with connected clients).

public int MaximumLU62Channels {get;}
The maximum number of channels that can be current, or clients that can be connected, that use the LU 6.2 transmission protocol.

public int MaximumMessageLength {get;}
Returns the maximum length of a message (in bytes) that can be handled by the queue manager. No queue can be defined with a maximum message length greater than `MaximumMessageLength`.

public int MaximumPriority {get;}
Returns the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to this value. Throws MQException if you call this method after disconnecting from the queue manager.

public int MaximumTCPChannels {get;}
The maximum number of channels that can be current, or clients that can be connected, that use the TCP/IP transmission protocol.

public int MQIAccounting {get;}
Controls the collection of accounting information for MQI data.

public int MQIStatistics {get;}
Controls the collection of statistics monitoring information for the queue manager.

public int OutboundPortMax {get;}
The maximum value in the range of port numbers to be used when binding outgoing channels.

public int OutboundPortMin {get;}
The minimum value in the range of port numbers to be used when binding outgoing channels.

public int QueueAccounting {get;}
Whether class 3 accounting (thread-level and queue-level accounting) data is to be used for all queues.

public int QueueMonitoring {get;}
Controls the collection of online monitoring data for queues.

public int QueueStatistics {get;}
Controls the collection of statistics data for queues.

public int ReceiveTimeout {get;}
The length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to the inactive state.

public int ReceiveTimeoutMin {get;}
The minimum length of time that a TCP/IP channel waits to receive data, including heartbeats, from its partner before returning to an inactive state.

public int ReceiveTimeoutType {get;}
The qualifier to apply to the value in `ReceiveTimeout`.

public int SharedQueueQueueManagerName {get;}
Specifies how to deliver messages to a shared queue. If the put specifies a different queue manager from the same queue sharing group as the target queue manager, the message is delivered in two ways:

MQC.MQSQQM_USE
Messages are delivered to the object queue manager before being put on the shared queue.

MQCMQSQQM_IGNORE
Messages are put directly on the shared queue.

public int SSLEvent {get;}
Whether SSL events are generated.

public int SSLFips {get;}
Whether only FIPS-certified algorithms are to be used if cryptography is performed in WebSphere MQ, rather than cryptographic hardware.

public int SSLKeyResetCount {get;}
Indicates the number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated.

public int ClusterSenderStatistics {get;}
Specifies the interval, in minutes, between consecutive gatherings of statistics.

public int SyncpointAvailability {get;}
Indicates whether the queue manager supports units of work and sync points with the MQQueue.get and MQQueue.put methods.

public string TCPName {get;}
The name of either the only, or default, TCP/IP system to be used, depending on the value of TCPStackType.

public int TCPStackType {get;}
Specifies whether the channel initiator uses only the TCP/IP address space specified in `TCPName`. Alternatively, the channel initiator can bind to any TCP/IP address.

public int TraceRouteRecording {get;}
Controls the recording of route tracing information.

Methods

public MQProcess AccessProcess(string *processName*, **int** *openOptions*);
public MQProcess AccessProcess(string *processName*, **int** *openOptions*, **string** *queueManagerName*, **string** *alternateUserId*);

Throws MQException.

Access a WebSphere MQ process on this queue manager to inquire on process attributes.

processName
The name of the process to open.

openOptions
Options that control the opening of the process. The valid options that can be added, or combined using a bitwise OR, are:

- `MQC.MQOO_FAIL_IF_QUIESCING`
- `MQC.MQOO_INQUIRE`
- `MQC.MQOO_SET`
- `MQC.MQOO_ALTERNATE_USER_AUTHORITY`

queueManagerName
The name of the queue manager on which the process is defined. You can leave a blank or null queue manager name if the queue manager is the same as the one the process is accessing.

alternateUserId

If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the *openOptions* parameter, *alternateUserId* specifies the alternative user ID used to check the authorization for the action. If MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be blank or null.

Default user authority is used for connection to the queue manager if MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified.

```
public MQQueue AccessQueue(string queueName, int openOptions);
public MQQueue AccessQueue(string queueName, int openOptions, string queueManagerName, string
dynamicQueueName, string alternateUserId);
```

Throws MQException.

Accesses a queue on this queue manager.

You can get or browse messages, put messages, inquire about the attributes of the queue or set the attributes of the queue. If the queue named is a model queue, a dynamic local queue is created. Query the name attribute of the resultant MQQueue object to find out the name of the dynamic queue.

queueName
Name of queue to open.

openOptions
Options that control the opening of the queue.

MQC.MQOO_ALTERNATE_USER_AUTHORITY
Validate with the specified user identifier.

MQC.MQOO_BIND_AS_QDEF
Use default binding for queue.

MQC.MQOO_BIND_NOT_FIXED
Do not bind to a specific destination.

MQC.MQOO_BIND_ON_OPEN
Bind handle to destination when queue is opened.

MQC.MQOO_BROWSE
Open to browse message.

MQC.MQOO_FAIL_IF_QUIESCING
Fail if the queue manager is quiescing.

MQC.MQOO_INPUT_AS_Q_DEF
Open to get messages using queue-defined default.

MQC.MQOO_INPUT_SHARED
Open to get messages with shared access.

MQC.MQOO_INPUT_EXCLUSIVE
Open to get messages with exclusive access.

MQC.MQOO_INQUIRE
Open for inquiry - required if you want to query properties.

MQC.MQOO_OUTPUT
Open to put messages.

MQC.MQOO_PASS_ALL_CONTEXT
Allow all context to be passed.

MQC.MQOO_PASS_IDENTITY_CONTEXT
Allow identity context to be passed.

MQC.MQOO_SAVE_ALL_CONTEXT
Save context when message retrieved.

MQC.MQOO_SET
Open to set attributes - required if you want to set properties.

MQC.MQOO_SET_ALL_CONTEXT
Allows all context to be set.

MQC.MQOO_SET_IDENTITY_CONTEXT
Allows identity context to be set.

queueManagerName
Name of the queue manager on which the queue is defined. A name that is entirely blank or null denotes the queue manager to which the MQQueueManager object is connected.

dynamicQueueName
dynamicQueueName is ignored unless queueName specifies the name of a model queue. If it does, *dynamicQueueName* specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last nonblank character in the name is an asterisk, *, the queue manager replaces the asterisk with a string of characters. The characters guarantee that the name generated for the queue is unique on this queue manager.

alternateUserId
If MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, *alternateUserId* specifies the alternate user identifier that is used to check the authorization for the open. If MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified, *alternateUserId* can be left blank, or null.

```
public MQTopic AccessTopic( MQDestination destination, string topicName,  string topicObject,
int options);
public MQTopic AccessTopic( MQDestination destination, string topicName,  string topicObject,
int options, string alternateUserId);
public MQTopic AccessTopic( MQDestination destination, string topicName,  string topicObject,
int options, string alternateUserId, string subscriptionName);
public MQTopic AccessTopic( MQDestination destination, string topicName,  string topicObject,
int options, string alternateUserId, string subscriptionName, System.Collections.Hashtable
properties);
public MQTopic AccessTopic(string topicName, string topicObject, int openAs, int options);
public MQTopic AccessTopic(string topicName, string topicObject, int openAs, int options, string
alternateUserId);
public MQTopic AccessTopic(string topicName, string topicObject, int options, string
alternateUserId, string subscriptionName);
public MQTopic AccessTopic(string topicName, string topicObject, int options, string
alternateUserId, string subscriptionName, System.Collections.Hashtable properties);
```

Access a topic on this queue manager.

MQTopic objects are closely related to administrative topic objects, which are sometimes called topic objects. On input, topicObject points to an administrative topic object. The MQTopic constructor obtains a topic string from the topic object and combines it with topicName to create a topic name. Either or bothtopicObject or topicName can be null. The topic name is matched to the topic tree, and the name of the closest matching administrative topic object is returned in topicObject.

The topics that are associated with the MQTopic object are the result of combining two topic strings. The first topic string is defined by the administrative topic object identified by *topicObject*. The second topic string is *topicString*. The resulting topic string associated with the MQTopic object can identify multiple topics by including wild cards.

Depending on whether the topic is opened for publishing or subscribing, you can use the MQTopic.Put methods to publish on topics, or MQTopic.Get methods to receive publications on topics. If you want to publish and subscribe to the same topic, you must access the topic twice, once for publish and once for subscribe.

If you create an MQTopic object for subscription, without providing an MQDestination object, a managed subscription is assumed. If you pass a queue as an MQDestination object, an unmanaged subscription is assumed. You must ensure the subscription options you set are consistent with the subscription being managed or unmanaged.

destination
: *destination* is an MQQueue instance. By providing *destination*, MQTopic is opened as an unmanaged subscription. Publications on the topic are delivered to the queue accessed as *destination*.

topicName
: A topic string that is the second part of the topic name. *topicName* is concatenated with the topic string defined in the *topicObject* administrative topic object. You can set *topicName* to null, in which case the topic name is defined by the topic string in *topicObject*.

topicObject
: On input, *topicObject* is the name of the topic object that contains the topic string that forms the first part of the topic name. The topic string in *topicObject* is concatenated with *topicName*. The rules for constructing topic names are defined in Combining topic strings.

  On output, *topicObject* contains the name of the administrative topic object that is the closest match in the topic tree to the topic identified by the topic name.

openAs
: Access the topic to publish or subscribe. The parameter can contain only one of these options:

  - MQC.MQTOPIC_OPEN_AS_SUBSCRIPTION
  - MQC.MQTOPIC_OPEN_AS_PUBLICATION

options

: Combine the options that control the opening of the topic for either publication or subscription. Use MQC.MQSO_* constants to access a topic for subscription and MQC.MQOO_* constants to access a topic for publication.

  If more than one option is required, add the values together, or combine the option values using the bitwise OR operator.

alternateUserId
: Specify the alternate user ID that is used to check for the required authorization to finish the operation. You must specify *alternateUserId*, if either MQC.MQOO_ALTERNATE_USER_AUTHORITY or MQC.MQSO_ALTERNATE_USER_AUTHORITY is set in the options parameter.

subscriptionName
: *subscriptionName* is required if the options MQC.MQSO_DURABLE or MQC.MQSO_ALTER are provided. In both cases, MQTopic is implicitly opened for subscription. An exception is thrown if the MQC.MQSO_DURABLE is set, and the subscription exists, or if MQC.MQSO_ALTER is set, and the subscription does not exist.

properties
: Set any of the special subscription properties listed using a hash table. Specified entries in the hash table are updated with output values. Entries are not added to the hash table to report output values.

  - MQC.MQSUB_PROP_ALTERNATE_SECURITY_ID
  - MQC.MQSUB_PROP_SUBSCRIPTION_EXPIRY
  - MQC.MQSUB_PROP_SUBSCRIPTION_USER_DATA

- MQC.MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQC.MQSUB_PROP_PUBLICATION_PRIORITY
- MQC.MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQC.MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

public MQAsyncStatus GetAsyncStatus();

Throws MQException

Returns an MQAsyncStatus object, which represents the asynchronous activity for the queue manager connection.

public void Backout();

Throws MQException.

Backout any messages that were read or written within sync point since the last sync point.

Messages that were written with the MQC.MQPMO_SYNCPOINT flag set are removed from queues. Messages read with the MQC.MQGMO_SYNCPOINT flag are reinstated on the queues they came from. If the messages are persistent, the changes are logged.

For reconnectable clients, the MQRC_NONE reason code is returned to a client after reconnection is successful.

public void Begin();

Throws MQException.

Begin is supported only in server bindings mode. It starts a global unit of work.

public void Commit();

Throws MQException.

Commit any messages that were read or written within sync point since the last sync point.

Messages written with the MQC.MQPMO_SYNCPOINT flag set are made available to other applications. Messages retrieved with the MQC.MQGMO_SYNCPOINT flag set are deleted. If the messages are persistent, the changes are logged.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if connection is lost while carrying out the commit call.
- MQRC_BACKED_OUT if the commit call is issued after reconnection.

Disconnect();

Throws MQException.

Close the connection to the queue manager. All objects accessed on this queue manager are not longer accessible to this application. To reaccess the objects, create a MQQueueManager object.

Generally, any work performed as part of a unit of work is committed. However, if the unit of work is managed by .NET, the unit of work might be rolled back.

```
public void Put(int type,  string destinationName, MQMessage message);
public void Put(int type,  string destinationName, MQMessage message MQPutMessageOptions
putMessageOptions);
public void Put(int type,  string destinationName, string queueManagerName, string topicString,
MQMessage message);
public void Put(string queueName, MQMessage message);
public void Put(string queueName, MQMessage message, MQPutMessageOptions putMessageOptions);
public void Put(string queueName, string queueManagerName, MQMessage message);
public void Put(string queueName, string queueManagerName, MQMessage message,
MQPutMessageOptions putMessageOptions);
```

public void Put(string *queueName*, **string** *queueManagerName*, **MQMessage** *message*, **MQPutMessageOptions** *putMessageOptions*, **string** *alternateUserId*);

Throws MQException.

Places a single message onto a queue or topic without creating an MQQueue or MQTopic object first.

queueName
The name of the queue onto which to place the message.

destinationName
The name of a destination object. It is either a queue or a topic depending on the value of *type*.

type
The type of destination object. You must not combine the options.

MQC.MQOT_Q
Queue

MQC.MQOT_TOPIC
Topic

queueManagerName

The name of the queue manager or queue manager alias, on which the queue is defined. If type MQC.MQOT_TOPIC is specified this parameter is ignored.

If the queue is a model queue, and the resolved queue manager name is not this queue manager, an MQException is thrown.

topicString

topicString is combined with the topic name in the *destinationName* topic object.

topicString is ignored if *destinationName* is a queue.

message
The message to send. Message is an input/output object.

The following reason codes are returned to a reconnectable client:

- MQRC_CALL_INTERRUPTED if the connection is broken while performing a Put call on a persistent message.
- MQRC_NONE if the connection is successful while performing a Put call on a non-persistent message (see Application Recovery).

putMessageOptions

Options controlling the actions of the put.

If you omit *putMessageOptions*, a default instance of *putMessageOptions* is created. *putMessageOptions* is an input/output object.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

alternateUserId

Specifies an alternate user identifier used to check authorization when placing the message on a queue.

You can omit *alternateUserId* if you do not set MQC.MQOO_ALTERNATE_USER_AUTHORITY in *putMessageOptions*. If you set MQC.MQOO_ALTERNATE_USER_AUTHORITY, you must also set *alternateUserId*. *alternateUserId* has not effect unless you also set MQC.MQOO_ALTERNATE_USER_AUTHORITY.

Constructors

```
public MQQueueManager();
public MQQueueManager(string queueManagerName);
public MQQueueManager(string queueManagerName, Int options);
public MQQueueManager(string queueManagerName, Int options, string channel, string connName);
public MQQueueManager(string queueManagerName, string channel, string connName);
public MQQueueManager(string queueManagerName, System.Collections.Hashtable properties);
```

Throws MQException.

Creates a connection to a queue manager. Select between creating a client connection or a server connection.

You must have inquire (inq) authority on the queue manager when attempting to connect to the queue manager. Without inquire authority, the connection attempt fails.

A client connection is created if one of the following conditions is true:

1. *channel* or *connName* are specified in the constructor.
2. *HostName*, *Port*, or *Channel* are specified in *properties*.
3. *MQEnvironment.HostName*, *MQEnvironment.Port*, or *MQEnvironment.Channel* are specified.

The values of the connection properties are defaulted in the order shown. The *channel* and *connName* in the constructor take precedence over the property values in the constructor. The constructor property values take precedence of the MQEnvironment properties.

The host name, channel name, and port are defined in the MQEnvironment class.

queueManagerName
Name of the queue manager, or queue manager group to connect to.

Omit the parameter, or leave it null, or blank to make a default queue manager selection. The default queue manager connection on a server is to the default queue manager on the server. The default queue manager connection on a client connection is to the queue manager the listener is connected to.

options
Specify MQCNO connection options. The values must be applicable to the type of connection being made. For example, if you specify the following server connection properties for a client connection an MQException is thrown.

- MQC.MQCNO_FASTPATH_BINDING
- MQC.MQCNO_STANDARD_BINDING

properties

The properties parameter takes a series of key/value pairs that override the properties set by MQEnvironment; see the example, "Override MQEnvironment properties" on page 2628. The following properties can be overridden:

- MQC.CONNECT_OPTIONS_PROPERTY
- MQC.CONNNAME_PROPERTY
- MQC.ENCRYPTION_POLICY_SUITE_B
- MQC.HOST_NAME_PROPERTY
- MQC.PORT_PROPERTY
- MQC.CHANNEL_PROPERTY
- MQC.SSL_CIPHER_SPEC_PROPERTY
- MQC.SSL_PEER_NAME_PROPERTY
- MQC.SSL_CERT_STORE_PROPERTY
- MQC.SSL_CRYPTO_HARDWARE_PROPERTY

- MQC.SECURITY_EXIT_PROPERTY
- MQC.SECURITY_USERDATA_PROPERTY
- MQC.SEND_EXIT_PROPERTY
- MQC.SEND_USERDATA_PROPERTY
- MQC.RECEIVE_EXIT_PROPERTY
- MQC.RECEIVE_USERDATA_PROPERTY
- MQC.USER_ID_PROPERTY
- MQC.PASSWORD_PROPERTY
- MQC.MQAIR_ARRAY
- MQC.KEY_RESET_COUNT
- MQC.FIPS_REQUIRED
- MQC.HDR_CMP_LIST
- MQC.MSG_CMP_LIST
- MQC.TRANSPORT_PROPERTY

channel
Name of a server connection channel

connName
Connection name in the format *HostName* (*Port*).

You can supply a list of *hostnames* and *ports* as an argument to the constructor MQQueueManager(String queueManagerName, Hashtable properties) using CONNECTION_NAME_PROPERTY.
For example:

```
ConnectionName = "fred.mq.com(2344),nick.mq.com(3746),tom.mq.com(4288)";
Hashtable Properties-new Hashtable();
properties.Add(MQC.CONNECTION_NAME_PROPERTY,ConnectionName);
MQQueueManager qmgr=new MQQueue Manager("qmgrname",properties);
```

When a connection attempt is made, the connection name list is processed in order. If the connection attempt to the first host name and port fails, then connection to the second pair of attributes is attempted. The client repeats this process until either a successful connection is made or the list is exhausted. If the list is exhausted, an appropriate reason code and completion code is returned to the client application.
When a port number is not provided for the connection name, the default port (configured in mqclient.ini) is used.

Set the Connection List

You can set the connection list by using the following methods when the automatic client reconnection options are set:

Set the connection list through MQSERVER
You can set the connection list through the command prompt.

In the command prompt, set

```
MQSERVER=SYSTEM.DEF.SVRCONN/TCP/Hostname1(Port1),Hostname2(Por2),Hostname3(Port3)
For Example:
MQSERVER=SYSTEM.DEF.SVRCONN/TCP/fred.mq.com(5266),nick.mq.com(6566),jack.mq.com(8413)
```

If you set the connection in the MQSERVER, do not set it in the application.

If you set the connection list in the application, the application overwrites whatever is set in the MQSERVER environment variable.

Set the connection list through the application

You can set the connection list in the application by specifying the host name and port properties.

```
String connName = "fred.mq.com(2344), nick.mq.com(3746), chris.mq.com(4288)";
MQQueueManager qm = new MQQueueManager("QM1", "TestChannel", connName);
```

Set the connection list through `app.config`

`App.config` is an XML file where you specify the key-value pairs.

In the connection list specify

```
<app.Settings>
<add key="Connection1" value="Hostname1(Port1)"/>
<add key="Connection2" value="Hostname2(Port2)"/>
<app.Settings>
```

For example:

```
<app.Settings>
<add key>="Connection1" value="fred.mq.com(2966)"/>
<add key>="Connection2" value="alex.mq.com(6533)"/>
<app.Settings>
```

You can directly change the connection list in the `app.config` file.

Set the connection list through MQEnvironment

To set the Connection list through the MQEnvironment, use the *ConnectionName* property.

```
MQEnvironment.ConnectionName = "fred.mq.com(4288),"alex.mq.com(5211);
```

The *ConnectionName* property overwrites the host name and port properties set in the MQEnvironment.

Create a client connection

The following example shows you how to create a client connection to a queue manager. You can create a client connection by setting the MQEnvironment variables before creating a new MQQueueManager Object.

```
MQEnvironment.Hostname = "fred.mq.com";  // host to connect to
MQEnvironment.Port     = 1414;           // port to connect to
                                         //If not explicitly set,
                                         // defaults to 1414
                                         // (the default WebSphere MQ port)
MQEnvironment.Channel  = "channel.name"; // the case sensitive
                                         //  name of the
                                         // SVR CONN channel on
                                         // the queue manager
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

Figure 52. Client connection

Override MQEnvironment properties

The following example shows you how to create a queue manager with its user ID and password defined in a hash table.

```
Hashtable properties = new Hashtable();

properties.Add( MQC.USER_ID_PROPERTY, "ExampleUserId" );
properties.Add( MQC.PASSWORD_PROPERTY, "ExamplePassword" );

try
{
    MQQueueManager qMgr = new MQQueueManager("qmgrname", properties);
}
catch (MQException mqe)
{
    System.Console.WriteLine("Connect failed with " + mqe.Message);
    return((int)mqe.Reason);
}
```

Figure 53. Overriding MQEnvironment properties

Create a reconnectable connection

The following example shows you how to automatically reconnect a client to a Queue Manager.

```
Hashtable properties = new Hashtable(); // The queue manager name and the
                                        // properties how it has to be connected

properties.Add(MQC.CONNECT_OPTIONS_PROPERTY, MQC.MQCNO_RECONNECT);//Options through which
                                           // through which reconnection happens

properties.Add(MQC.CONNECTION_NAME_PROPERTY,"fred.mq.com(4789),nick.mq.com(4790)");// The list of
                                           // queue managers through which reconnect happens

MQ QueueManager qmgr = new MQQueueManager("qmgrname", properties);
```

Figure 54. Automatically reconnecting a client to a queue manager

MQSubscription .NET class

Use MQSubscription to request that retained publications are sent to the subscriber. MQSubscription is a property of an MQTopic object opened for subscription.

Class

```
System.Object
  |
  └─ IBM.WMQ.MQBase
     |
     └─ IBM.WMQ.MQBaseObject
        |
        └─ IBM.WMQ.MQManagedObject
           |
           └─ IBM.WMQ.MQSubscription
```

```
public class IBM.WMQ.MQSubscription extends IBM.WMQ.MQManagedObject;
```

- "Properties"
- "Methods" on page 2630
- "Constructors" on page 2630

Properties

Access subscription properties using the MQManagedObject class; see "Properties" on page 2589.

Methods

Access subscription Inquire, Set and Get methods using the MQManagedObject class; see "Methods" on page 2590.

public int RequestPublicationUpdate(int *options***);**

Throws MQException.

Request an updated publication for the current topic. If the queue manager has a retained publications for the topic, they are sent to the subscriber.

Before calling RequestPublicationUpdate, open a topic for subscription to obtain an MQSubscription object.

Typically, open the subscription with the MQC.MQSO_PUBLICATIONS_ON_REQUEST option. If no wildcards are present in the topic string, then only one publication is sent as a result of this call. If the topic string contains wildcards, many publications might be sent. The method returns the number of retained publications that are sent to the subscription queue. There is no guarantee that this many publications are received, especially if they are non-persistent messages.

options

MQC.MQSRO_FAIL_IF_QUIESCING
The method fails if the queue manager is in a quiescent state. On z/OS, for a CICS or IMS application, MQC.MQSRO_FAIL_IF_QUIESCING also forces the method to fail if the connection is in a quiescent state.

MQC.MQSRO_NONE
No options are specified.

Constructors

No Public constructor.

An MQSubscription object is returned in the SubscriptionReference property of an MQTopic object that is opened for subscription,

Call the RequestPublicationUpdate method. MQSubscription is a subclass of MQManagedObject. Use the reference to access the properties and methods of MQManagedObject.

MQTopic .NET class

Use MQTopic to publish or subscribe messages on a topic, or to query or set attributes of a topic. Create an MQTopic object for publishing or subscribing by using a constructor or the MQQueueManager.AccessTopic method.

Class

```
System.Object
  |
  +- IBM.WMQ.MQBase
    |
    +- IBM.WMQ.MQBaseObject
      |
      +- IBM.WMQ.MQManagedObject
        |
        +- IBM.WMQ.MQDestination
          |
          +- IBM.WMQ.MQTopic
```

```
public class IBM.WMQ.MQTopic extends IBM.WMQ.MQDestination;
```

- "Properties" on page 2631

- "Methods"
- "Constructors" on page 2633

Properties

Test for MQException being thrown when getting properties.

public Boolean IsDurable {get;}
Read only property that returns `True` if the subscription is durable or `False` otherwise. If the topic was opened for publication, the property is ignored and would always return `False`.

public Boolean IsManaged {get;};
Read only property that returns `True` if the subscription is managed by the queue manager, or `False` otherwise. If the topic was opened for publication, the property is ignored and would always return False.

public Boolean IsSubscribed {get;};
Read only property that returns `True` if the topic was opened for subscription and `False` if the topic was opened for publication.

public MQSubscription SubscriptionReference {get;};
Read only property that returns the MQSubscription object associated with a topic object opened for subscription. The reference is available if you want to modify the close options or start any of the objects methods.

public MQDestination UnmanagedDestinationReference {get;};
Read only property that returns the MQQueue associated with an unmanaged subscription. It is the destination specified when the topic object was created. The property returns null for any topic objects opened for publication or with a managed subscription.

Methods

public void Put(MQMessage *message***);**
public void Put(MQMessage *message***, MQPutMessageOptions** *putMessageOptions***);**
Throws MQException.

Publishes a message to the topic.

Modifications to the MQMessage object after the Put call has been accomplished do not affect the actual message on the WebSphere MQ queue or publication topic.

Put updates the `MessageId` and `CorrelationId` properties of the MQMessage object and does not clear message data. Further Put or Get calls refer to the updated information in the MQMessage object. For example, in the following code snippet, the first message contains `a` and the second `ab`.

```
msg.WriteString("a");
q.Put(msg,pmo);
msg.WriteString("b");
q.Put(msg,pmo);
```

message
An MQMessage object containing the message descriptor data, and message to be sent. The message descriptor can be altered as a consequence of this method. The values in the message descriptor immediately after the completion of this method are the values that were put to the queue or published to the topic.

The following reason codes are returned to a reconnectable client:

- `MQRC_CALL_INTERRUPTED` if the connection is broken while running a Put call on a persistent message and the reconnection is successful.
- `MQRC_NONE` if the connection is successful while running a Put call on a non-persistent message (see Application Recovery).

putMessageOptions
Options controlling the action of the put.

If *putMessageOptions* is not specified the default instance of MQPutMessageOptons is used.

If you use the MQPMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

Note: For simplicity and performance, if you want to put a single message to a queue, use MQQueueManager.Put object. You should have an MQQueue object for this.

```
public void Get(MQMessage message);
public void Get(MQMessage message, MQGetMessageOptions getMessageOptions);
public void Get(MQMessage message, MQGetMessageOptions getMessageOptions, int MaxMsgSize);
```
Throws MQException.

Retrieves a message from the topic.

This method uses a default instance of MQGetMessageOptions to do the get. The message option used is MQGMO_NOWAIT.

If the get fails, the MQMessage object is unchanged. If it succeeds, the message descriptor and message data portions of the MQMessage are replaced with the message descriptor and message data from the incoming message.

All calls to WebSphere MQ from a particular MQQueueManager are synchronous. Therefore, if you perform a get with wait, all other threads using the same MQQueueManager are blocked from making further WebSphere MQ calls until the Get call is accomplished. If you need multiple threads to access WebSphere MQ simultaneously, each thread must create its own MQQueueManager object.

message
Contains the message descriptor and the returned message data. Some of the fields in the message descriptor are input parameters. It is important to ensure that the MessageId and CorrelationId input parameters are set as required.

A reconnectable client returns the reason code MQRC_BACKED_OUT after successful reconnection, for messages received under MQGM_SYNCPOINT.

getMessageOptions
Options controlling the action of the get.

Using option MQC.MQGMO_CONVERT might result in an exception with reason code MQC.MQRC_CONVERTED_STRING_TOO_BIG when converting from single-byte character codes to double byte codes. In this case, the message is copied into the buffer without conversion.

If *getMessageOptions* is not specified, the message option used is MQGMO_NOWAIT.

If you use the MQGMO_LOGICAL_ORDER option in a reconnectable client, the MQRC_RECONNECT_INCOMPATIBLE reason code is returned.

MaxMsgSize
The largest message this message object is to receive. If the message on the queue is larger than this size, one of two things occurs:

- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is set in the MQGetMessageOptions object, the message is filled with as much of the message data as possible. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_ACCEPTED reason code.
- If the MQGMO_ACCEPT_TRUNCATED_MSG flag is not set, the message is left on the queue. An exception is thrown with the MQCC_WARNING completion code and MQRC_TRUNCATED_MSG_FAILED reason code.

If *MaxMsgSize* is not specified, the whole message is retrieved.

Constructors

```
public MQTopic(MQQueueManager queueManager, MQDestination destination, string topicName, string
topicObject, int options);
public MQTopic(MQQueueManager queueManager, MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId);
public MQTopic(MQQueueManager queueManager, MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId, string subscriptionName);
public MQTopic(MQQueueManager queueManager, MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId, string subscriptionName,
System.Collections.Hashtable properties);
public MQTopic(MQQueueManager queueManager, string topicName, string topicObject, int openAs,
int options);
public MQTopic(MQQueueManager queueManager, string topicName, string topicObject, int openAs,
int options, string alternateUserId);
public MQTopic(MQQueueManager queueManager, string topicName, string topicObject, int options,
string alternateUserId, string subscriptionName);
public MQTopic(MQQueueManager queueManager, string topicName, string topicObject, int options,
string alternateUserId, string subscriptionName, System.Collections.Hashtable properties);
```

Access a topic on *queueManager*.

MQTopic objects are closely related to administrative topic objects, which are sometimes called topic objects. On input, `topicObject` points to an administrative topic object. The MQTopic constructor obtains a topic string from the topic object and combines it with `topicName` to create a topic name. Either or both`topicObject` or `topicName` can be null. The topic name is matched to the topic tree, and the name of the closest matching administrative topic object is returned in `topicObject`.

The topics that are associated with the MQTopic object are the result of combining two topic strings. The first topic string is defined by the administrative topic object identified by *topicObject*. The second topic string is *topicString*. The resulting topic string associated with the MQTopic object can identify multiple topics by including wild cards.

Depending on whether the topic is opened for publishing or subscribing, you can use the MQTopic.Put methods to publish on topics, or MQTopic.Get methods to receive publications on topics. If you want to publish and subscribe to the same topic, you must access the topic twice, once for publish and once for subscribe.

If you create an MQTopic object for subscription, without providing an MQDestination object, a managed subscription is assumed. If you pass a queue as an MQDestination object, an unmanaged subscription is assumed. You must ensure the subscription options you set are consistent with the subscription being managed or unmanaged.

queueManager
Queue manager to access a topic on.

destination
destination is an MQQueue instance. By providing *destination*, MQTopic is opened as an unmanaged subscription. Publications on the topic are delivered to the queue accessed as *destination*.

topicName
A topic string that is the second part of the topic name. *topicName* is concatenated with the topic string defined in the *topicObject* administrative topic object. You can set *topicName* to null, in which case the topic name is defined by the topic string in *topicObject*.

topicObject
On input, *topicObject* is the name of the topic object that contains the topic string that forms the first part of the topic name. The topic string in *topicObject* is concatenated with *topicName*. The rules for constructing topic names are defined in Combining topic strings.

On output, *topicObject* contains the name of the administrative topic object that is the closest match in the topic tree to the topic identified by the topic name.

openAs
Access the topic to publish or subscribe. The parameter can contain only one of these options:

- MQC.MQTOPIC_OPEN_AS_SUBSCRIPTION
- MQC.MQTOPIC_OPEN_AS_PUBLICATION

options

Combine the options that control the opening of the topic for either publication or subscription. Use MQC.MQSO_* constants to access a topic for subscription and MQC.MQOO_* constants to access a topic for publication.

If more than one option is required, add the values together, or combine the option values using the bitwise OR operator.

alternateUserId
Specify the alternate user ID that is used to check for the required authorization to finish the operation. You must specify *alternateUserId*, if either MQC.MQOO_ALTERNATE_USER_AUTHORITY or MQC.MQSO_ALTERNATE_USER_AUTHORITY is set in the options parameter.

subscriptionName
subscriptionName is required if the options MQC.MQSO_DURABLE or MQC.MQSO_ALTER are provided. In both cases, MQTopic is implicitly opened for subscription. An exception is thrown if the MQC.MQSO_DURABLE is set, and the subscription exists, or if MQC.MQSO_ALTER is set, and the subscription does not exist.

properties
Set any of the special subscription properties listed using a hash table. Specified entries in the hash table are updated with output values. Entries are not added to the hash table to report output values.

- MQC.MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQC.MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQC.MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQC.MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQC.MQSUB_PROP_PUBLICATION_PRIORITY
- MQC.MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQC.MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

```
public MQTopic MQQueueManager.AccessTopic(MQDestination destination, string topicName, string
topicObject, int options);
public MQTopic MQQueueManager.AccessTopic(MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId);
public MQTopic MQQueueManager.AccessTopic(MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId, string subscriptionName);
public MQTopic MQQueueManager.AccessTopic(MQDestination destination, string topicName, string
topicObject, int options, string alternateUserId, string subscriptionName,
System.Collections.Hashtable properties);
public MQTopic MQQueueManager.AccessTopic(string topicName, string topicObject, int openAs, int
options);
public MQTopic MQQueueManager.AccessTopic(string topicName, string topicObject, int openAs, int
options, string alternateUserId);
public MQTopic MQQueueManager.AccessTopic(string topicName, string topicObject, int options,
string alternateUserId, string subscriptionName);
public MQTopic MQQueueManager.AccessTopic(string topicName, string topicObject, int options,
string alternateUserId, string subscriptionName, System.Collections.Hashtable properties);
```

Access a topic on this queue manager.

MQTopic objects are closely related to administrative topic objects, which are sometimes called topic objects. On input, `topicObject` points to an administrative topic object. The MQTopic constructor obtains a topic string from the topic object and combines it with `topicName` to create a topic name. Either or both`topicObject` or `topicName` can be null. The topic name is matched to the topic tree, and the name of the closest matching administrative topic object is returned in `topicObject`.

The topics that are associated with the MQTopic object are the result of combining two topic strings. The first topic string is defined by the administrative topic object identified by *`topicObject`*. The second topic string is *`topicString`*. The resulting topic string associated with the MQTopic object can identify multiple topics by including wild cards.

Depending on whether the topic is opened for publishing or subscribing, you can use the MQTopic.Put methods to publish on topics, or MQTopic.Get methods to receive publications on topics. If you want to publish and subscribe to the same topic, you must access the topic twice, once for publish and once for subscribe.

If you create an MQTopic object for subscription, without providing an MQDestination object, a managed subscription is assumed. If you pass a queue as an MQDestination object, an unmanaged subscription is assumed. You must ensure the subscription options you set are consistent with the subscription being managed or unmanaged.

`destination`
: *`destination`* is an MQQueue instance. By providing *`destination`*, MQTopic is opened as an unmanaged subscription. Publications on the topic are delivered to the queue accessed as *`destination`*.

`topicName`
: A topic string that is the second part of the topic name. *`topicName`* is concatenated with the topic string defined in the *`topicObject`* administrative topic object. You can set *`topicName`* to null, in which case the topic name is defined by the topic string in *`topicObject`*.

`topicObject`
: On input, *`topicObject`* is the name of the topic object that contains the topic string that forms the first part of the topic name. The topic string in *`topicObject`* is concatenated with *`topicName`*. The rules for constructing topic names are defined in Combining topic strings.

  On output, *`topicObject`* contains the name of the administrative topic object that is the closest match in the topic tree to the topic identified by the topic name.

`openAs`
: Access the topic to publish or subscribe. The parameter can contain only one of these options:
  - `MQC.MQTOPIC_OPEN_AS_SUBSCRIPTION`
  - `MQC.MQTOPIC_OPEN_AS_PUBLICATION`

`options`

: Combine the options that control the opening of the topic for either publication or subscription. Use `MQC.MQSO_*` constants to access a topic for subscription and `MQC.MQOO_*` constants to access a topic for publication.

  If more than one option is required, add the values together, or combine the option values using the bitwise `OR` operator.

`alternateUserId`
: Specify the alternate user ID that is used to check for the required authorization to finish the operation. You must specify *`alternateUserId`*, if either `MQC.MQOO_ALTERNATE_USER_AUTHORITY` or `MQC.MQSO_ALTERNATE_USER_AUTHORITY` is set in the options parameter.

subscriptionName
subscriptionName is required if the options MQC.MQSO_DURABLE or MQC.MQSO_ALTER are provided. In both cases, MQTopic is implicitly opened for subscription. An exception is thrown if the MQC.MQSO_DURABLE is set, and the subscription exists, or if MQC.MQSO_ALTER is set, and the subscription does not exist.

properties
Set any of the special subscription properties listed using a hash table. Specified entries in the hash table are updated with output values. Entries are not added to the hash table to report output values.

- MQC.MQSUB_PROP_ALTERNATE_SECURITY_ID
- MQC.MQSUB_PROP_SUBSCRIPTION_EXPIRY
- MQC.MQSUB_PROP_SUBSCRIPTION_USER_DATA
- MQC.MQSUB_PROP_SUBSCRIPTION_CORRELATION_ID
- MQC.MQSUB_PROP_PUBLICATION_PRIORITY
- MQC.MQSUB_PROP_PUBLICATION_ACCOUNTING_TOKEN
- MQC.MQSUB_PROP_PUBLICATION_APPLICATIONID_DATA

IMQObjectTrigger .NET interface

Implement IMQObjectTrigger to process messages passed by the **runmqdnm** .NET monitor.

Interface

```
public interface IBM.WMQMonitor.IMQObjectTrigger();
```

Depending on whether sync point control is specified in the **runmqdnm** command the message is removed from the queue before or after the Execute method returns.

Methods

void Execute (MQQueueManager *queueManager*, **MQQueue** *queue*, **MQMessage** *message*, **string** *param*);

queueManager
Queue manager hosting the queue being monitored.

queue
Queue being monitored.

message
Message read from the queue.

param
Data passed from UserParameter.

MQC .NET interface

Refer to an MQI constant by prefixing the constant name with MQC.. MQC defines all the constants used by the MQI.

Interface

```
System.Object
  |
  └─ IBM.WMQ.MQC

public interface IBM.WMQ.MQC extends System.Object;
```

Example

```
MQQueue queue;
queue.closeOptions = MQC.MQCO_DELETE;
```

Character set identifiers for .NET applications

Descriptions of the character sets you can select to encode .NET WebSphere MQ messages

Character set	Description
37	ibm037
437	ibm437 / PC Original
500	ibm500
819	iso-8859-1 / latin1 / ibm819
1200	Unicode
1208	UTF-8
273	ibm273
277	ibm277
278	ibm278
280	ibm280
284	ibm284
285	ibm285
297	ibm297
420	ibm420
424	ibm424
737	ibm737 / PC Greek
775	ibm775 / PC Baltic
813	iso-8859-7 / greek / ibm813
838	ibm838
850	ibm850 / PC Latin 1
852	ibm852 / PC Latin 2
855	ibm855 / PC Cyrillic
856	ibm856
857	ibm857 / PC Turkish
860	ibm860 / PC Portuguese
861	ibm861 / PC Icelandic
862	ibm862 / PC Hebrew
863	ibm863 / PC Canadian French
864	ibm864 / PC Arabic
865	ibm865 / PC Nordic
866	ibm866 / PC Russian

Character set	Description
868	ibm868
869	ibm869 / PC Modern Greek
870	ibm870
871	ibm871
874	ibm874
875	ibm875
912	iso-8859-2 / latin2 / ibm912
913	iso-8859-3 / latin3 / ibm913
914	iso-8859-4 / latin4 / ibm914
915	iso-8859-5 / cyrillic / ibm915
916	iso-8859-8 / hebrew / ibm916
918	ibm918
920	iso-8859-9 / latin5 / ibm920
921	ibm921
922	ibm922
930	ibm930
932	PC Japanese
933	ibm933
935	ibm935
937	ibm937
939	ibm939
942	ibm942
943	ibm943
948	ibm948
949	ibm949
950	ibm950 / Big 5 Traditional Chinese
954	EUCJIS
964	ibm964 / CNS 11643 Traditional Chinese
970	ibm970
1006	ibm1006
1025	ibm1025
1026	ibm1026
1089	iso-8859-6 / arabic / ibm1089
1097	ibm1097
1098	ibm1098
1112	ibm1112
1122	ibm1122
1123	ibm1123
1124	ibm1124
1250	Windows Latin 2
1251	Windows Cyrillic

Character set	Description
1252	Windows Latin 1
1253	Windows Greek
1254	Windows Turkish
1255	Windows Hebrew
1256	Windows Arabic
1257	Windows Baltic
1258	Windows Vietnamese
1381	ibm1381
1383	ibm1383
2022	JIS
5601	ksc-5601 Korean
33722	ibm33722

WebSphere MQ C++ classes

The WebSphere MQ C++ classes encapsulate the WebSphere MQ Message Queue Interface (MQI). There is a single C++ header file, **imqi.hpp**, which covers all of these classes.

For each class, the following information is shown:

Class hierarchy diagram
A class diagram showing the class in its inheritance relation to its immediate parent classes, if any.

Other relevant classes
Document links to other relevant classes, such as parent classes, and the classes of objects used in method signatures.

Object attributes
Attributes of the class. These are in addition to those attributes defined for any parent classes. Many attributes reflect WebSphere MQ data-structure members (see "C++ and MQI cross-reference" on page 2641). For detailed descriptions, see "Attributes of objects" on page 2096.

Constructors
Signatures of the special methods used to create an object of the class.

Object methods (public)
Signatures of methods that require an instance of the class for their operation, and that have no usage restrictions.

Where it applies, the following information is also shown:

Class methods (public)
Signatures of methods that do not require an instance of the class for their operation, and that have no usage restrictions.

Overloaded (parent class) methods
Signatures of those virtual methods that are defined in parent classes, but exhibit different, polymorphic, behavior for this class.

Object methods (protected)
Signatures of methods that require an instance of the class for their operation, and are reserved for use by the implementations of derived classes. This section is of interest only to class writers, as opposed to class users.

Object data (protected)
Implementation details for object instance data available to the implementations of derived classes. This section is of interest only to class writers, as opposed to class users.

Reason codes
MQRC_* values (see API reason codes) that can be expected from those methods that fail. For an exhaustive list of reason codes that can occur for an object of a class, consult the parent class documentation. The documented list of reason codes for a class does not include the reason codes for parent classes.

Note:

1. Objects of these classes are not thread-safe. This ensures optimal performance, but take care not to access any object from more than one thread.
2. It is recommended that, for a multithreaded program, a separate ImqQueueManager object is used for each thread. Each manager object must have its own independent collection of other objects, ensuring that objects in different threads are isolated from one another.

The classes are:

C++ and MQI cross-reference

This collection of topics contains information relating C++ to the MQI.

Read this information together with "Data types used in the MQI" on page 1529.

This table relates MQI data structures to the C++ classes and include files. The following topics show cross-reference information for each C++ class. These cross-references relate to the use of the underlying WebSphere MQ procedural interfaces. The classes ImqBinary, ImqDistributionList, and ImqString have no attributes that fall into this category and are excluded.

Table 257. Data structure, class, and include-file cross reference

Data structure	Class	Include file
MQAIR	ImqAuthenticationRecord	imqair.hpp
	ImqBinary	imqbin.hpp
	ImqCache	imqcac.hpp
MQCD	ImqChannel	imqchl.hpp
MQCIH	ImqCICSBridgeHeader	imqcih.hpp
MQDLH	ImqDeadLetterHeader	imqdlh.hpp
MQOR	ImqDistributionList	imqdst.hpp
	ImqError	imqerr.hpp
MQGMO	ImqGetMessageOptions	imqgmo.hpp
	ImqHeader	imqhdr.hpp
MQIIH	ImqIMSBridgeHeader	imqiih.hpp
	ImqItem	imqitm.hpp
MQMD	ImqMessage	imqmsg.hpp
	ImqMessageTracker	imqmtr.hpp
	ImqNamelist	imqnml.hpp
MQOD, MQRR	ImqObject	imqobj.hpp
MQPMO, MQPMR, MQRR	ImqPutMessageOptions	imqpmo.hpp
	ImqProcess	imqpro.hpp
	ImqQueue	imqque.hpp
MQBO, MQCNO, MQCSP	ImqQueueManager	imqmgr.hpp
MQRMH	ImqReferenceHeader	imqrfh.hpp
	ImqString	imqstr.hpp
MQTM	ImqTrigger	imqtrg.hpp
MQTMC		
MQTMC2	ImqTrigger	imqtrg.hpp
MQXQH		
MQWIH	ImqWorkHeader	imqwih.hpp

ImqAuthenticationRecord cross-reference:

Cross-reference of attributes, data structures, fields, and calls for the ImqAuthenticationRecord C++ class.

Attribute	Data structure	Field	Call
connection name	MQAIR	AuthInfoConnName	MQCONNX
password	MQAIR	LDAPPassword	MQCONNX
type	MQAIR	AuthInfoType	MQCONNX
user name	MQAIR	LDAPUserNamePtr	MQCONNX
	MQAIR	LDAPUserNameOffset	MQCONNX
	MQAIR	LDAPUserNameLength	MQCONNX

ImqCache cross-reference:

Cross-reference of attributes and calls for the ImqCache C++ class.

Attribute	Call
automatic buffer	MQGET
buffer length	MQGET
buffer pointer	MQGET, MQPUT
data length	MQGET
data offset	MQGET
data pointer	MQGET
message length	MQGET, MQPUT

ImqChannel cross-reference:

Cross-reference of attributes, data structures, fields, and calls for the ImqChannel C++ class.

Attribute	Data structure	Field	Call
batch heart-beat	MQCD	BatchHeartbeat	MQCONNX
channel name	MQCD	ChannelName	MQCONNX
connection name	MQCD	ConnectionName	MQCONNX
	MQCD	ShortConnectionName	MQCONNX
header compression	MQCD	HdrCompList	MQCONNX
heart-beat interval	MQCD	HeartbeatInterval	MQCONNX
keep alive interval	MQCD	KeepAliveInterval	MQCONNX
local address	MQCD	LocalAddress	MQCONNX
maximum message length	MQCD	MaxMsgLength	MQCONNX
message compression	MQCD	MsgCompList	MQCONNX
mode name	MQCD	ModeName	MQCONNX
password	MQCD	Password	MQCONNX
receive exit count	MQCD		MQCONNX
receive exit names	MQCD	ReceiveExit	MQCONNX
	MQCD	ReceiveExitsDefined	MQCONNX

Attribute	Data structure	Field	Call
	MQCD	ReceiveExitPtr	MQCONNX
receive user data	MQCD	ReceiveUserData	MQCONNX
	MQCD	ReceiveUserDataPtr	MQCONNX
security exit name	MQCD	SecurityExit	MQCONNX
security user data	MQCD	SecurityUserData	MQCONNX
send exit count	MQCD		MQCONNX
send exit names	MQCD	SendExit	MQCONNX
	MQCD	SendExitsDefined	MQCONNX
	MQCD	SendExitPtr	MQCONNX
send user data	MQCD	SendUserData	MQCONNX
	MQCD	SendUserDataPtr	MQCONNX
SSL CipherSpec	MQCD	sslCipherSpecification	MQCONNX
SSL client authentication type	MQCD	sslClientAuthentication	MQCONNX
SSL peer name	MQCD	sslPeerName	MQCONNX
transaction program name	MQCD	TpName	MQCONNX
transport type	MQCD	TransportType	MQCONNX
user id	MQCD	UserIdentifier	MQCONNX

ImqCICSBridgeHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqCICSBridgeHeader C++ class.

Attribute	Data structure	Field
bridge abend code	MQCIH	AbendCode
ADS descriptor	MQCIH	AdsDescriptor
attention identifier	MQCIH	AttentionId
authenticator	MQCIH	Authenticator
bridge completion code	MQCIH	BridgeCompletionCode
bridge error offset	MQCIH	ErrorOffset
bridge reason code	MQCIH	BridgeReason
bridge cancel code	MQCIH	CancelCode
conversational task	MQCIH	ConversationalTask
cursor position	MQCIH	CursorPosition
facility token	MQCIH	Facility
facility keep time	MQCIH	FacilityKeepTime
facility like	MQCIH	FacilityLike
function	MQCIH	Function
get wait interval	MQCIH	GetWaitInterval
link type	MQCIH	LinkType
next transaction identifier	MQCIH	NextTransactionId
output data length	MQCIH	OutputDataLength
reply-to format	MQCIH	ReplyToFormat

Attribute	Data structure	Field
bridge return code	MQCIH	ReturnCode
start code	MQCIH	StartCode
task end status	MQCIH	TaskEndStatus
transaction identifier	MQCIH	TransactionId
uow control	MQCIH	UowControl
version	MQCIH	Version

ImqDeadLetterHeader cross reference:

Cross-reference of attributes, data structures, and fields for the ImqDeadLetterHeader C++ class.

Attribute	Data structure	Field
dead-letter reason code	MQDLH	Reason
destination queue manager name	MQDLH	DestQMgrName
destination queue name	MQDLH	DestQName
put application name	MQDLH	PutApplName
put application type	MQDLH	PutApplType
put date	MQDLH	PutDate
put time	MQDLH	PutTime

ImqError cross reference:

Cross-reference of attributes and calls for the ImqError C++ class.

Attribute	Call
completion code	MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQSET
reason code	MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX, MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQSET

ImqGetMessageOptions cross reference:

Cross-reference of attributes, data structures, and fields for the ImqGetMessageOptions C++ class.

Attribute	Data structure	Field
group status	MQGMO	GroupStatus
match options	MQGMO	MatchOptions
message token	MQGMO	MessageToken
options	MQGMO	Options
resolved queue name	MQGMO	ResolvedQName
returned length	MQGMO	ReturnedLength
segmentation	MQGMO	Segmentation
segment status	MQGMO	SegmentStatus
	MQGMO	Signal1

Attribute	Data structure	Field
	MQGMO	Signal2
syncpoint participation	MQGMO	Options
wait interval	MQGMO	WaitInterval

ImqHeader cross reference:

Cross-reference of attributes, data structures, and fields for the ImqHeader C++ class.

Attribute	Data structure	Field
character set	MQDLH, MQIIH	CodedCharSetId
encoding	MQDLH, MQIIH	Encoding
format	MQDLH, MQIIH	Format
header flags	MQIIH, MQRMH	Flags

ImqIMSBridgeHeader cross reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Attribute	Data structure	Field
authenticator	MQIIH	Authenticator
commit mode	MQIIH	CommitMode
logical terminal override	MQIIH	LTermOverride
message format services map name	MQIIH	MFSMapName
reply-to format	MQIIH	ReplyToFormat
security scope	MQIIH	SecurityScope
transaction instance id	MQIIH	TranInstanceId
transaction state	MQIIH	TranState

ImqItem cross reference:

Cross-reference of attributes and calls for the ImqItem C++ class.

Attribute	Call
structure id	MQGET

ImqMessage cross reference:

Cross-reference of attributes, data structures, fields, and calls for the ImqMessage C++ class.

Attribute	Data structure	Field	Call
application id data	MQMD	ApplIdentityData	
application origin data	MQMD	ApplOriginData	
backout count	MQMD	BackoutCount	
character set	MQMD	CodedCharSetId	
encoding	MQMD	Encoding	
expiry	MQMD	Expiry	
format	MQMD	Format	
message flags	MQMD	MsgFlags	
message type	MQMD	MsgType	
offset	MQMD	Offset	
original length	MQMD	OriginalLength	
persistence	MQMD	Persistence	
priority	MQMD	Priority	
put application name	MQMD	PutApplName	
put application type	MQMD	PutApplType	
put date	MQMD	PutDate	
put time	MQMD	PutTime	
reply-to queue manager name	MQMD	ReplyToQMgr	
reply-to queue name	MQMD	ReplyToQ	
report	MQMD	Report	
sequence number	MQMD	MsgSeqNumber	
total message length		DataLength	MQGET
user id	MQMD	UserIdentifier	

ImqMessageTracker cross reference:

Cross-reference of attributes, data structures, and fields for the ImqMessageTracker C++ class.

Attribute	Data structure	Field
accounting token	MQMD	AccountingToken
correlation id	MQMD	CorrelId
feedback	MQMD	Feedback
group id	MQMD	GroupId
message id	MQMD	MsgId

ImqNamelist cross reference:

Cross-reference of attributes, inquiries, and calls for the ImqNamelist C++ class.

Attribute	Inquiry	Call
name count	MQIA_NAME_COUNT	MQINQ
namelist name	MQCA_NAMELIST_NAME	MQINQ

ImqObject cross reference:

Cross-reference of attributes, data structures, fields, inquiries, and calls for the ImqObject C++ class.

Attribute	Data structure	Field	Inquiry	Call
alteration date			MQCA_ALTERATION_DATE	MQINQ
alteration time			MQCA_ALTERATION_TIME	MQINQ
alternate user id	MQOD	AlternateUserId		
alternate security id				
close options				MQCLOSE
description			MQCA_Q_DESC, MQCA_Q_MGR_DESC, MQCA_PROCESS_DESC	MQINQ
name	MQOD	ObjectName	MQCA_Q_MGR_NAME, MQCQ_Q_NAME, MQCA_PROCESS_NAME	MQINQ
open options				MQOPEN
open status				MQOPEN, MQCLOSE
queue manager identifier	queue manager identifier		MQCA_Q_MGR_IDENTIFIER	MQINQ

ImqProcess cross-reference:

Cross-reference of attributes, inquiries, and calls for the ImqAuthenticationRecord C++ class.

Attribute	Inquiry	Call
application id	MQCA_APPL_ID	MQINQ
application type	MQIA_APPL_TYPE	MQINQ
environment data	MQCA_ENV_DATA	MQINQ
user data	MQCA_USER_DATA	MQINQ

ImqPutMessageOptions cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Table 258. ImqPutMessageOptions cross reference

Attribute	Data structure	Field
context reference	MQPMO	Context
	MQPMO	InvalidDestCount
	MQPMO	KnownDestCount
options	MQPMO	Options
record fields	MQPMO	PutMsgRecFields
resolved queue manager name	MQPMO	ResolvedQMgrName
resolved queue name	MQPMO	ResolvedQName
	MQPMO	Timeout
	MQPMO	UnknownDestCount
syncpoint participation	MQPMO	Options

ImqQueue cross-reference:

Cross-reference of attributes, data structures, fields, inquiries, and calls for the ImqQueue C++ class.

Table 259. ImqQueue cross reference

Attribute	Data structure	Field	Inquiry	Call
backout requeue name			MQCA_BACKOUT_REQ_Q_NAME	MQINQ
backout threshold			MQIA_BACKOUT_THRESHOLD	MQINQ
base queue name			MQCA_BASE_Q_NAME	MQINQ
cluster name			MQCA_CLUSTER_NAME	MQINQ
cluster namelist name			MQCA_CLUSTER_NAMELIST	MQINQ
cluster workload rank			MQIA_CLWL_Q_RANK	MQINQ
cluster workload priority			MQIA_CLWL_Q_PRIORITY	MQINQ
cluster workload use queue			MQIA_CLWL_USEQ	MQINQ
creation date			MQCA_CREATION_DATE	MQINQ
creation time			MQCA_CREATION_TIME	MQINQ
current depth			MQIA_CURRENT_Q_DEPTH	MQINQ
default bind			MQIA_DEF_BIND	MQINQ
default input open option			MQIA_DEF_INPUT_OPEN_OPTION	MQINQ
default persistence			MQIA_DEF_PERSISTENCE	MQINQ
default priority			MQIA_DEF_PRIORITY	MQINQ
definition type			MQIA_DEFINITION_TYPE	MQINQ
depth high event			MQIA_Q_DEPTH_HIGH_EVENT	MQINQ
depth high limit			MQIA_Q_DEPTH_HIGH_LIMIT	MQINQ
depth low event			MQIA_Q_DEPTH_LOW_EVENT	MQINQ

Table 259. ImqQueue cross reference (continued)

Attribute	Data structure	Field	Inquiry	Call
depth low limit			MQIA_Q_DEPTH_LOW_LIMIT	MQINQ
depth maximum event			MQIA_Q_DEPTH_MAX_LIMIT	MQINQ
distribution lists			MQIA_DIST_LISTS	MQINQ, MQSET
dynamic queue name	MQOD	DynamicQName		
harden get backout			MQIA_HARDEN_GET_BACKOUT	MQINQ
index type			MQIA_INDEX_TYPE	MQINQ
inhibit get			MQIA_INHIBIT_GET	MQINQ, MQSET
inhibit put			MQIA_INHIBIT_PUT	MQINQ, MQSET
initiation queue name			MQCA_INITIATION_Q_NAME	MQINQ
maximum depth			MQIA_MAX_Q_DEPTH	MQINQ
maximum message length			MQIA_MAX_MSG_LENGTH	MQINQ
message delivery sequence			MQIA_MSG_DELIVERY_SEQUENCE	MQINQ
next distributed queue				
non persistent message class			MQIA_NPM_CLASS	MQINQ
open input count			MQIA_OPEN_INPUT_COUNT	MQINQ
open output count			MQIA_OPEN_OUTPUT_COUNT	MQINQ
previous distributed queue				
process name			MQCA_PROCESS_NAME	MQINQ
queue accounting			MQIA_ACCOUNTING_Q	MQINQ
queue manager name	MQOD	ObjectQMgrName		
queue monitoring			MQIA_MONITORING_Q	MQINQ
queue statistics			MQIA_STATISTICS_Q	MQINQ
queue type			MQIA_Q_TYPE	MQINQ
remote queue manager name			MQCA_REMOTE_Q_MGR_NAME	MQINQ
remote queue name			MQCA_REMOTE_Q_NAME	MQINQ
resolved queue manager name	MQOD	ResolvedQMgrName		
resolved queue name	MQOD	ResolvedQName		
retention interval			MQIA_RETENTION_INTERVAL	MQINQ
scope			MQIA_SCOPE	MQINQ
service interval			MQIA_Q_SERVICE_INTERVAL	MQINQ
service interval event			MQIA_Q_SERVICE_INTERVAL_EVENT	MQINQ
shareability			MQIA_SHAREABILITY	MQINQ
storage class			MQCA_STORAGE_CLASS	MQINQ

Table 259. ImqQueue cross reference (continued)

Attribute	Data structure	Field	Inquiry	Call
transmission queue name			MQCA_XMIT_Q_NAME	MQINQ
trigger control			MQIA_TRIGGER_CONTROL	MQINQ, MQSET
trigger data			MQCA_TRIGGER_DATA	MQINQ, MQSET
trigger depth			MQIA_TRIGGER_DEPTH	MQINQ, MQSET
trigger message priority			MQIA_TRIGGER_MSG_PRIORITY	MQINQ, MQSET
trigger type			MQIA_TRIGGER_TYPE	MQINQ, MQSET
usage			MQIA_USAGE	MQINQ

ImqQueueManager cross-reference:

Cross-reference of attributes, data structures, fields, inquiries, and calls for the ImqQueueManager C++ class.

Attribute	Data structure	Field	Inquiry	Call
accounting connections override			MQIA_ACCOUNTING_CONN_OVERRIDE	MQINQ
accounting interval			MQIA_ACCOUNTING_INTERVAL	MQINQ
activity recording			MQIA_ACTIVITY_RECORDING	MQINQ
adopt new mca check			MQIA_ADOPTNEWMCA_CHECK	MQINQ
adopt new mca type			MQIA_ADOPTNEWMCA_TYPE	MQINQ
authentication type	MQCSP	AuthenticationType		MQCONNX
authority event			MQIA_AUTHORITY_EVENT	MQINQ
begin options	MQBO	Options		MQBEGIN
bridge event			MQIA_BRIDGE_EVENT	MQINQ
channel auto definition			MQIA_CHANNEL_AUTO_DEF	MQINQ
channel auto definition event			MQIA_CHANNEL_AUTO_EVENT	MQIA
channel auto definition exit			MQIA_CHANNEL_AUTO_EXIT	MQIA
channel event			MQIA_CHANNEL_EVENT	MQINQ
channel initiator adapters			MQIA_CHINIT_ADAPTERS	MQINQ
channel initiator control			MQIA_CHINIT_CONTROL	MQINQ
channel initiator dispatchers			MQIA_CHINIT_DISPATCHERS	MQINQ

Attribute	Data structure	Field	Inquiry	Call
channel initiator trace auto start			MQIA_CHINIT_TRACE_AUTO_START	MQINQ
channel initiator trace table size			MQIA_CHINIT_TRACE_TABLE_SIZE	MQINQ
channel monitoring			MQIA_MONITORING_CHANNEL	MQINQ
channel reference	MQCD	ChannelType		MQCONNX
channel statistics			MQIA_STATISTICS_CHANNEL	MQINQ
character set			MQIA_CODED_CHAR_SET_ID	MQINQ
cluster sender monitoring			MQIA_MONITORING_AUTO_CLUSSDR	MQINQ
cluster sender statistics			MQIA_STATISTICS_AUTO_CLUSSDR	MQINQ
cluster workload data			MQCA_CLUSTER_WORKLOAD_DATA	MQINQ
cluster workload exit			MQCA_CLUSTER_WORKLOAD_EXIT	MQINQ
cluster workload length			MQIA_CLUSTER_WORKLOAD_LENGTH	MQINQ
cluster workload mru			MQIA_CLWL_MRU_CHANNELS	MQINQ
cluster workload use queue			MQIA_CLWL_USEQ	MQINQ
command event			MQIA_COMMAND_EVENT	MQINQ
command input queue name			MQCA_COMMAND_INPUT_Q_NAME	MQINQ
command level			MQIA_COMMAND_LEVEL	MQINQ
command server control			MQIA_CMD_SERVER_CONTROL	MQINQ
connect options	MQCNO	Options		MQCONN, MQCONNX
connection id	MQCNO	ConnectionId		MQCONNX
connection status				MQCONN, MQCONNX, MQDISC
connection tag	MQCD	ConnTag		MQCONNX
cryptographic hardware	MQSCO	CryptoHardware		MQCONNX
dead-letter queue name			MQCA_DEAD_LETTER_Q_NAME	MQINQ
default transmission queue name			MQCA_DEF_XMIT_Q_NAME	MQINQ
distribution lists			MQIA_DIST_LISTS	MQINQ
dns group			MQCA_DNS_GROUP	MQINQ
dns wlm			MQIA_DNS_WLM	MQINQ
first authentication record	MQSCO	AuthInfoRecOffset		MQCONNX

Attribute	Data structure	Field	Inquiry	Call
	MQSCO	AuthInfoRecPtr		MQCONNX
inhibit event			MQIA_INHIBIT_EVENT	MQINQ
ip address version			MQIA_IP_ADDRESS_VERSION	MQINQ
key repository	MQSCO	KeyRepository		MQCONNX
key reset count	MQSCO	KeyResetCount		MQCONNX
listener timer			MQIA_LISTENER_TIMER	MQINQ
local event			MQIA_LOCAL_EVENT	MQINQ
logger event			MQIA_LOGGER_EVENT	MQINQ
lu group name			MQCA_LU_GROUP_NAME	MQINQ
lu name			MQCA_LU_NAME	MQINQ
lu62 arm suffix			MQCA_LU62_ARM_SUFFIX	MQINQ
lu62 channels			MQIA_LU62_CHANNELS	MQINQ
maximum active channels			MQIA_ACTIVE_CHANNELS	MQINQ
maximum channels			MQIA_MAX_CHANNELS	MQINQ
maximum handles			MQIA_MAX_HANDLES	MQINQ
maximum message length			MQIA_MAX_MSG_LENGTH	MQINQ
maximum priority			MQIA_MAX_PRIORITY	MQINQ
maximum uncommitted messages			MQIA_MAX_UNCOMMITTED_MSGS	MQINQ
mqi accounting			MQIA_ACCOUNTING_MQI	MQINQ
mqi statistics			MQIA_STATISTICS_MQI	MQINQ
outbound port maximum			MQIA_OUTBOUND_PORT_MAX	MQINQ
outbound port minimum			MQIA_OUTBOUND_PORT_MIN	MQINQ
password	MQCSP	CSPPasswordPtr		MQCONNX
	MQCSP	CSPPasswordOffset		MQCONNX
	MQCSP	CSPPasswordLength		MQCONNX
performance event			MQIA_PERFORMANCE_EVENT	MQINQ
platform			MQIA_PLATFORM	MQINQ
queue accounting			MQIA_ACCOUNTING_Q	MQINQ
queue monitoring			MQIA_MONITORING_Q	MQINQ
queue statistics			MQIA_STATISTICS_Q	MQINQ
receive timeout			MQIA_RECEIVE_TIMEOUT	MQINQ
receive timeout minimum			MQIA_RECEIVE_TIMEOUT_MIN	MQINQ
receive timeout type			MQIA_RECEIVE_TIMEOUT_TYPE	MQINQ
remote event			MQIA_REMOTE_EVENT	MQINQ
repository name			MQCA_REPOSITORY_NAME	MQINQ

Attribute	Data structure	Field	Inquiry	Call
repository namelist			MQCA_REPOSITORY_NAMELIST	MQINQ
shared queue queue manager name			MQIA_SHARED_Q_Q_MGR_NAME	MQINQ
ssl event			MQIA_SSL_EVENT	MQINQ
ssl fips			MQIA_SSL_FIPS_REQUIRED	MQINQ
ssl key reset count			MQIA_SSL_RESET_COUNT	MQINQ
start-stop event			MQIA_START_STOP_EVENT	MQINQ
statistics interval			MQIA_STATISTICS_INTERVAL	MQINQ
syncpoint availability			MQIA_SYNCPOINT	MQINQ
tcp channels			MQIA_TCP_CHANNELS	MQINQ
tcp keep alive			MQIA_TCP_KEEP_ALIVE	MQINQ
tcp name			MQCA_TCP_NAME	MQINQ
tcp stack type			MQIA_TCP_STACK_TYPE	MQINQ
trace route recording			MQIA_TRACE_ROUTE_RECORDING	MQINQ
trigger interval			MQIA_TRIGGER_INTERVAL	MQINQ
user id	MQCSP	CSPUserIdPtr		MQCONNX
	MQCSP	CSPUserIdOffset		MQCONNX
	MQCSP	CSPUserIdLength		MQCONNX

ImqReferenceHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Attribute	Data structure	Field
destination environment	MQRMH	DestEnvLength, DestEnvOffset
destination name	MQRMH	DestNameLength, DestNameOffset
instance id	MQRMH	ObjectInstanceId
logical length	MQRMH	DataLogicalLength
logical offset	MQRMH	DataLogicalOffset
logical offset 2	MQRMH	DataLogicalOffset2
reference type	MQRMH	ObjectType
source environment	MQRMH	SrcEnvLength, SrcEnvOffset
source name	MQRMH	SrcNameLength, SrcNameOffset

ImqTrigger cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Table 260. ImqTrigger cross reference

Attribute	Data structure	Field
application id	MQTM	ApplId
application type	MQTM	ApplType
environment data	MQTM	EnvData
process name	MQTM	ProcessName
queue name	MQTM	QName
trigger data	MQTM	TriggerData
user data	MQTM	UserData

ImqWorkHeader cross-reference:

Cross-reference of attributes, data structures, and fields for the ImqAuthenticationRecord C++ class.

Attribute	Data structure	Field
message token	MQWIH	MessageToken
service name	MQWIH	ServiceName
service step	MQWIH	ServiceStep

ImqAuthenticationRecord C++ class

This class encapsulates an authentication information record (MQAIR) for use during execution of the ImqQueueManager::connect method, for custom SSL client connections.

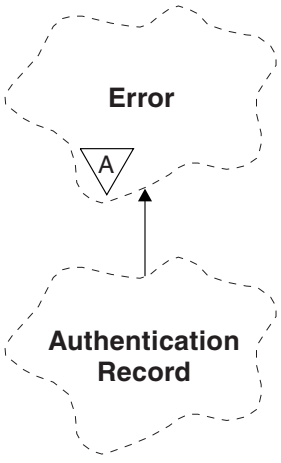


Figure 55. ImqAuthenticationRecord class

See the description of the ImqQueueManager::connect method for more details. This class is not available on the z/OS platform.

- "Object attributes" on page 2655
- "Constructors" on page 2655
- "Object methods (public)" on page 2655

• "Object methods (protected)" on page 2656

Object attributes

connection name
The name of the connection to the LDAP CRL server. This is the IP address or DNS name, followed optionally by the port number, in parentheses.

connection reference
A reference to an ImqQueueManager object that provides the required connection to a (local) queue manager. The initial value is zero. Do not confuse this with the queue manager name that identifies a queue manager (possibly remote) for a named queue.

next authentication record
Next object of this class, in no particular order, having the same **connection reference** as this object. The initial value is zero.

password
A password supplied for connection authentication to the LDAP CRL server.

previous authentication record
Previous object of this class, in no particular order, having the same **connection reference** as this object. The initial value is zero.

type The type of authentication information contained in the record.

user name
A user identifier supplied for authorization to the LDAP CRL server.

Constructors

ImqAuthenticationRecord( );
The default constructor.

Object methods (public)

void operator = (const ImqAuthenticationRecord & *air***);**
Copies instance data from *air*, replacing the existing instance data.

const ImqString & connectionName ( ) const ;
Returns the **connection name**.

void setConnectionName (const ImqString & *name* **);**
Sets the **connection name**.

void setConnectionName (const char * *name* **= 0);**
Sets the **connection name**.

ImqQueueManager * connectionReference ( ) const ;
Returns the **connection reference**.

void setConnectionReference (ImqQueueManager & *manager* **);**
Sets the **connection reference**.

void setConnectionReference (ImqQueueManager * *manager* **= 0);**
Sets the **connection reference**.

void copyOut (MQAIR * *pAir* **);**
Copies instance data to *pAir*, replacing the existing instance data. This might involve allocating dependent storage.

void clear (MQAIR * *pAir* **);**
Clears the structure and releases dependent storage referenced by *pAir*.

ImqAuthenticationRecord * nextAuthenticationRecord ( ) const ;
Returns the **next authentication record**.

const ImqString & password ( ) const ;
Returns the **password**.

void setPassword (const ImqString & *password* **);**
Sets the **password**.

void setPassword (const char * *password* **= 0);**
Sets the **password**.

ImqAuthenticationRecord * previousAuthenticationRecord ( ) const ;
Returns the **previous authentication record**.

MQLONG type ( ) const ;
Returns the **type**.

void setType (const MQLONG *type* **);**
Sets the **type**.

const ImqString & userName ( ) const ;
Returns the **user name**.

void setUserName (const ImqString & *name* **);**
Sets the **user name**.

void setUserName (const char * *name* **= 0);**
Sets the **user name**.

Object methods (protected)

void setNextAuthenticationRecord (ImqAuthenticationRecord * *pAir* **= 0);**
Sets the **next authentication record**.

Attention: Use this function only if you are sure that it will not break the authentication record list.

void setPreviousAuthenticationRecord (ImqAuthenticationRecord * *pAir* **= 0);**
Sets the **previous authentication record**.

Attention: Use this function only if you are sure that it will not break the authentication record list.

ImqBinary C++ class

This class encapsulates a binary byte array that can be used for ImqMessage **accounting token**, **correlation id**, and **message id** values. It allows easy assignment, copying, and comparison.

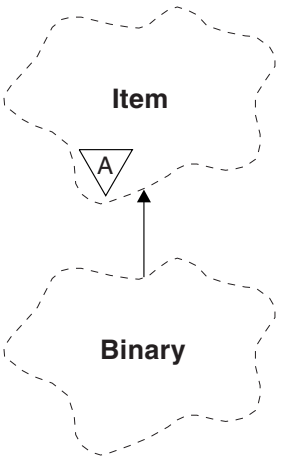


Figure 56. ImqBinary class

Object attributes

data An array of bytes of binary data. The initial value is null.

data length
The number of bytes. The initial value is zero.

data pointer
The address of the first byte of the **data**. The initial value is zero.

Constructors

ImqBinary( );
The default constructor.

ImqBinary(const ImqBinary & *binary* **);**
The copy constructor.

ImqBinary(const void * *data*, **const size_t** *length* **);**
Copies *length* bytes from *data*.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg* **);**
Copies the **data** to the message buffer, replacing any existing content. Sets the *msg* **format** to MQFMT_NONE.

See the ImqItem class method description for further details.

virtual ImqBoolean pasteIn(ImqMessage & *msg* **);**
Sets the **data** by transferring the remaining data from the message buffer, replacing the existing **data**.

To be successful, the ImqMessage **format** must be MQFMT_NONE.

See the ImqItem class method description for further details.

Object methods (public)

void operator = (const ImqBinary & *binary*);
Copies bytes from *binary*.

ImqBoolean operator == (const ImqBinary & *binary*);
Compares this object with *binary*. It returns FALSE if not equal and TRUE otherwise. The objects are equal if they have the same **data length** and the bytes match.

ImqBoolean copyOut(void * *buffer*, const size_t *length*, const char *pad* = 0);
Copies up to *length* bytes from the **data pointer** to *buffer*. If the **data length** is insufficient, the remaining space in *buffer* is filled with *pad* bytes. *buffer* can be zero if *length* is also zero. *length* must not be negative. It returns TRUE if successful.

size_t dataLength( ) const ;
Returns the **data length**.

ImqBoolean setDataLength(const size_t *length*);
Sets the **data length**. If the **data length** is changed as a result of this method, the data in the object is uninitialized. It returns TRUE if successful.

void * dataPointer( ) const ;
Returns the **data pointer**.

ImqBoolean isNull( ) const ;
Returns TRUE if the **data length** is zero, or if all the **data** bytes are zero. Otherwise it returns FALSE.

ImqBoolean set(const void * *buffer*, const size_t *length*);
Copies *length* bytes from *buffer*. It returns TRUE if successful.

Object methods (protected)

void clear( );
Reduces the **data length** to zero.

Reason codes

- MQRC_NO_BUFFER
- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_INCONSISTENT_FORMAT

ImqCache C++ class

Use this class to hold or marshal data in memory.

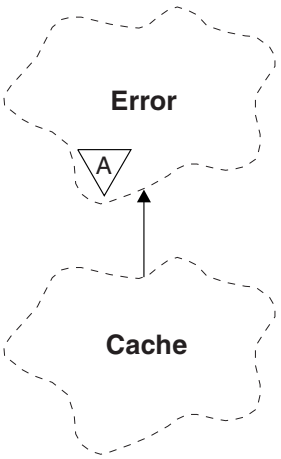


Figure 57. ImqCache class

Use this class to hold or marshal data in memory. You can nominate a buffer of memory of fixed size, or the system can provide a flexible amount of memory automatically. This class relates to the MQI calls listed in "ImqCache cross-reference" on page 2642.

Object attributes

automatic buffer
Indicates whether buffer memory is managed automatically by the system (TRUE) or is supplied by the user (FALSE). It is initially set to TRUE.

This attribute is not set directly. It is set indirectly using either the **useEmptyBuffer** or the **useFullBuffer** method.

If user storage is supplied, this attribute is FALSE, buffer memory cannot grow, and buffer overflow errors can occur. The address and length of the buffer remain constant.

If user storage is not supplied, this attribute is TRUE, and buffer memory can grow incrementally to accommodate an arbitrary amount of message data. However, when the buffer grows, the address of the buffer might change, so be careful when using the **buffer pointer** and **data pointer**.

buffer length
The number of bytes of memory in the buffer. The initial value is zero.

buffer pointer
The address of the buffer memory. The initial value is null.

data length
The number of bytes succeeding the **data pointer**. This must be equal to or less than the **message length**. The initial value is zero.

data offset
The number of bytes preceding the **data pointer**. This must be equal to or less than the **message length**. The initial value is zero.

data pointer
The address of the part of the buffer that is to be written to or read from next. The initial value is null.

message length
The number of bytes of significant data in the buffer. The initial value is zero.

Constructors

ImqCache( );
The default constructor.

ImqCache(const ImqCache & *cache* **);**
The copy constructor.

Object methods (public)

void operator = (const ImqCache & *cache* **);**
Copies up to **message length** bytes of data from the *cache* object to the object. If **automatic buffer** is FALSE, the **buffer length** must already be sufficient to accommodate the copied data.

ImqBoolean automaticBuffer( ) const ;
Returns the **automatic buffer** value.

size_t bufferLength( ) const ;
Returns the **buffer length**.

char * bufferPointer( ) const ;
Returns the **buffer pointer**.

void clearMessage( );
Sets the **message length** and **data offset** to zero.

size_t dataLength( ) const ;
Returns the **data length**.

size_t dataOffset( ) const ;
Returns the **data offset**.

ImqBoolean setDataOffset(const size_t *offset* **);**
Sets the **data offset**. The **message length** is increased if necessary to ensure that it is no less than the **data offset**. This method returns TRUE if successful.

char * dataPointer( ) const ;
Returns a copy of the **data pointer**.

size_t messageLength( ) const ;
Returns the **message length**.

ImqBoolean setMessageLength(const size_t *length* **);**
Sets the **message length**. Increases the **buffer length** if necessary to ensure that the **message length** is no greater than the **buffer length**. Reduces the **data offset** if necessary to ensure that it is no greater than the **message length**. It returns TRUE if successful.

ImqBoolean moreBytes(const size_t *bytes-required* **);**
Assures that *bytes-required* more bytes are available (for writing) between the **data pointer** and the end of the buffer. It returns TRUE if successful.

If **automatic buffer** is TRUE, more memory is acquired as required; otherwise, the **buffer length** must already be adequate.

ImqBoolean read(const size_t *length***, char * &** *external-buffer* **);**
Copies *length* bytes, from the buffer starting at the **data pointer** position, into the *external-buffer*. After the data has been copied, the **data offset** is increased by *length*. This method returns TRUE if successful.

ImqBoolean resizeBuffer(const size_t *length* **);**
Varies the **buffer length**, provided that **automatic buffer** is TRUE. This is achieved by

reallocating the buffer memory. Up to **message length** bytes of data from the existing buffer are copied to the new one. The maximum number copied is *length* bytes. The **buffer pointer** is changed. The **message length** and **data offset** are preserved as closely as possible within the confines of the new buffer. It returns TRUE if successful, and FALSE if **automatic buffer** is FALSE.

Note: This method can fail with MQRC_STORAGE_NOT_AVAILABLE if there is any problem with system resources.

ImqBoolean useEmptyBuffer(const char * *external-buffer*, **const size_t** *length* **);**
Identifies an empty user buffer, setting the **buffer pointer** to point to *external-buffer*, the **buffer length** to *length*, and the **message length** to zero. Performs a **clearMessage**. If the buffer is fully primed with data, use the **useFullBuffer** method instead. If the buffer is partially primed with data, use the **setMessageLength** method to indicate the correct amount. This method returns TRUE if successful.

This method can be used to identify a fixed amount of memory, as described previously (*external-buffer* is not null and *length* is nonzero), in which case **automatic buffer** is set to FALSE, or it can be used to revert to system-managed flexible memory (*external-buffer* is null and *length* is zero), in which case **automatic buffer** is set to TRUE.

ImqBoolean useFullBuffer(const char * *externalBuffer*, **const size_t** *length* **);**
As for **useEmptyBuffer**, except that the **message length** is set to *length*. It returns TRUE if successful.

ImqBoolean write(const size_t *length*, **const char *** *external-buffer* **);**
Copies *length* bytes, from the *external-buffer*, into the buffer starting at the **data pointer** position. After the data has been copied, the **data offset** is increased by *length*, and the **message length** is increased if necessary to ensure that it is no less than the new **data offset** value. This method returns TRUE if successful.

If **automatic buffer** is TRUE, an adequate amount of memory is guaranteed; otherwise, the ultimate **data offset** must not exceed the **buffer length**.

Reason codes

- MQRC_BUFFER_NOT_AUTOMATIC
- MQRC_DATA_TRUNCATED
- MQRC_INSUFFICIENT_BUFFER
- MQRC_INSUFFICIENT_DATA
- MQRC_NULL_POINTER
- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_ZERO_LENGTH

ImqChannel C++ class

This class encapsulates a channel definition (MQCD) for use during execution of the Manager::connect method, for custom client connections.

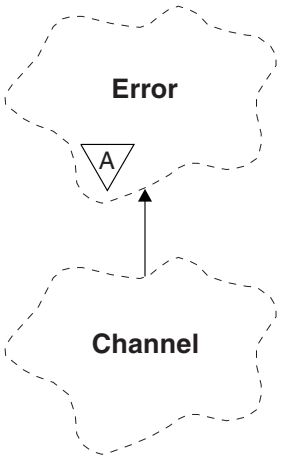


Figure 58. ImqChannel class

See the description of the Manager::connect method, and Sample program HELLO WORLD (imqwrld.cpp), for more details. Not all the listed methods are applicable to all platforms; see the descriptions of the DEFINE CHANNEL and ALTER CHANNEL commands in MQSC reference for more details. The ImqChannel class is not supported on z/OS.

Object attributes

batch heart-beat
The number of milliseconds between checks that a remote channel is active. The initial value is 0.

channel name
The name of the channel. The initial value is null.

connection name
The name of the connection. For example, the IP address of a host computer. The initial value is null.

header compression
The list of header data compression techniques supported by the channel. The initial values are all set to MQCOMPRESS_NOT_AVAILABLE.

heart-beat interval
The number of seconds between checks that a connection is still working. The initial value is 300.

keep alive interval
The number of seconds passed to the communications stack specifying the keep alive timing for the channel. The initial value is MQKAI_AUTO.

local address
The local communications address for the channel.

maximum message length
The maximum length of message supported by the channel in a single communication. The initial value is 4 194 304.

message compression
The list of message data compression techniques supported by the channel. The initial values are all set to MQCOMPRESS_NOT_AVAILABLE.

mode name
The name of the mode. The initial value is null.

password
A password supplied for connection authentication. The initial value is null.

receive exit count
The number of receive exits. The initial value is zero. This attribute is read-only.

receive exit names
The names of receive exits.

receive user data
Data associated with receive exits.

security exit name
The name of a security exit to be invoked on the server side of the connection. The initial value is null.

security user data
Data to be passed to the security exit. The initial value is null.

send exit count
The number of send exits. The initial value is zero. This attribute is read-only.

send exit names
The names of send exits.

send user data
Data associated with send exits.

SSL CipherSpec
CipherSpec for use with SSL.

SSL client authentication type
Client authentication type for use with SSL.

SSL peer name
Peer name for use with SSL.

transaction program name
The name of the transaction program. The initial value is null.

transport type
The transport type of the connection. The initial value is MQXPT_LU62.

user id
A user identifier supplied for authorization. The initial value is null.

Constructors

ImqChannel( ) ;
The default constructor.

ImqChannel(const ImqChannel & *channel* **);**
The copy constructor.

Object methods (public)

void operator = (const ImqChannel & *channel* **);**
Copies instance data from *channel*, replacing any existing instance data.

MQLONG batchHeartBeat( ) const ;
Returns the **batch heart-beat**.

ImqBoolean setBatchHeartBeat(const MQLONG *heartbeat* **= 0L);**
Sets the **batch heart-beat** . This method returns TRUE if successful.

ImqString channelName( ) const ;
Returns the **channel name**.

ImqBoolean setChannelName(const char * *name* **= 0);**
Sets the **channel name**. This method returns TRUE if successful.

ImqString connectionName( ) const ;
Returns the **connection name**.

ImqBoolean setConnectionName(const char * *name* **= 0);**
Sets the **connection name**. This method returns TRUE if successful.

size_t headerCompressionCount( ) const ;
Returns the supported header data compression techniques count.

ImqBoolean headerCompression(const size_t count, MQLONG compress [ ]) const ;
Returns copies of the supported header data compression techniques in **compress**. This method returns TRUE if successful.

ImqBoolean setHeaderCompression(const size_t count, const MQLONG compress [ ]);

Sets the supported header data compression techniques to **compress**.

Sets the supported header data compression techniques count to **count**.

This method returns TRUE if successful.

MQLONG heartBeatInterval( ) const ;
Returns the **heart-beat interval**.

ImqBoolean setHeartBeatInterval(const MQLONG *interval* **= 300L);**
Sets the **heart-beat interval**. This method returns TRUE if successful.

MQLONG keepAliveInterval( ) const ;
Returns the **keep alive interval**.

ImqBoolean setKeepAliveInterval(const MQLONG *interval* **= MQKAI_AUTO);**
Sets the **keep alive interval**. This method returns TRUE if successful.

ImqString localAddress() const ;
Returns the **local address**.

ImqBoolean setLocalAddress (const char * *address* **= 0);**
Sets the **local address**. This method returns TRUE if successful.

MQLONG maximumMessageLength( ) const ;
Returns the **maximum message length**.

ImqBoolean setMaximumMessageLength(const MQLONG *length* **= 4194304L);**
Sets the **maximum message length**. This method returns TRUE if successful.

size_t messageCompressionCount( ) const ;
Returns the supported message data compression techniques count.

ImqBoolean messageCompression(const size_t count, MQLONG compress [ ]) const ;
Returns copies of the supported message data compression techniques in **compress**. This method returns TRUE if successful.

ImqBoolean setMessageCompression(const size_t count, const MQLONG compress [ ]);

Sets the supported message data compression techniques to compress.

Sets the supported message data compression techniques count to count.

This method returns TRUE if successful.

ImqString modeName( ) const ;
Returns the **mode name**.

ImqBoolean setModeName(const char * *name* = 0);
Sets the **mode name**. This method returns TRUE if successful.

ImqString password( ) const ;
Returns the **password**.

ImqBoolean setPassword(const char * *password* = 0);
Sets the **password**. This method returns TRUE if successful.

size_t receiveExitCount( ) const ;
Returns the **receive exit count**.

ImqString receiveExitName( );
Returns the first of the **receive exit names**, if any. If the **receive exit count** is zero, it returns an empty string.

ImqBoolean receiveExitNames(const size_t *count*, ImqString * *names* [ ]);
Returns copies of the **receive exit names** in *names*. Sets any *names* in excess of **receive exit count** to null strings. This method returns TRUE if successful.

ImqBoolean setReceiveExitName(const char * *name* = 0);
Sets the **receive exit names** to the single *name*. *name* can be blank or null. Sets the **receive exit count** to either 1 or zero. Clears the **receive user data**. This method returns TRUE if successful.

ImqBoolean setReceiveExitNames(const size_t *count*, const char * *names* [ ]);
Sets the **receive exit names** to *names*. Individual *names* values must not be blank or null. Sets the **receive exit count** to *count*. Clears the **receive user data**. This method returns TRUE if successful.

ImqBoolean setReceiveExitNames(const size_t *count*, const ImqString * *names* [ ]);
Sets the **receive exit names** to *names*. Individual *names* values must not be blank or null. Sets the **receive exit count** to *count*. Clears the **receive user data**. This method returns TRUE if successful.

ImqString receiveUserData( );
Returns the first of the **receive user data** items, if any. If the **receive exit count** is zero, returns an empty string.

ImqBoolean receiveUserData(const size_t *count*, ImqString * *data* [ ]);
Returns copies of the **receive user data** items in *data*. Sets any *data* in excess of **receive exit count** to null strings. This method returns TRUE if successful.

ImqBoolean setReceiveUserData(const char * *data* = 0);
Sets the **receive user data** to the single item *data*. If *data* is not null, **receive exit count** must be at least 1. This method returns TRUE if successful.

ImqBoolean setReceiveUserData(const size_t *count*, const char * *data* [ ]);
Sets the **receive user data** to *data*. *count* must be no greater than the **receive exit count**. This method returns TRUE if successful.

ImqBoolean setReceiveUserData(const size_t *count*, const ImqString * *data* [ ]);
Sets the **receive user data** to *data*. *count* must be no greater than the **receive exit count**. This method returns TRUE if successful.

ImqString securityExitName( ) const ;
Returns the **security exit name**.

ImqBoolean setSecurityExitName(const char * *name* = 0);
Sets the **security exit name**. This method returns TRUE if successful.

ImqString securityUserData( ) const ;
Returns the **security user data**.

ImqBoolean setSecurityUserData(const char * *data* **= 0);**
Sets the **security user data**. This method returns TRUE if successful.

size_t sendExitCount( ) const ;
Returns the **send exit count**.

ImqString sendExitName( );
Returns the first of the **send exit names**, if any. Returns an empty string if the **send exit count** is zero.

ImqBoolean sendExitNames(const size_t *count*, **ImqString** * *names* **[ ]);**
Returns copies of the **send exit names** in *names*. Sets any *names* in excess of **send exit count** to null strings. This method returns TRUE if successful.

ImqBoolean setSendExitName(const char * *name* **= 0);**
Sets the **send exit names** to the single *name*. *name* can be blank or null. Sets the **send exit count** to either 1 or zero. Clears the **send user data**. This method returns TRUE if successful

ImqBoolean setSendExitNames(const size_t *count*, **const char** * *names* **[ ]);**
Sets the **send exit names** to *names*. Individual *names* values must not be blank or null. Sets the **send exit count** to *count*. Clears the **send user data**. This method returns TRUE if successful.

ImqBoolean setSendExitNames(const size_t *count*, **const ImqString** * *names* **[ ]);**
Sets the **send exit names** to *names*. Individual *names* values must not be blank or null. Sets the **send exit count** to *count*. Clears the **send user data**. This method returns TRUE if successful.

ImqString sendUserData( );
Returns the first of the **send user data** items, if any., Returns an empty string if the **send exit count** is zero.

ImqBoolean sendUserData(const size_t *count*, **ImqString** * *data* **[ ]);**
Returns copies of the **send user data** items in *data*. Sets any *data* in excess of **send exit count** to null strings. This method returns TRUE if successful.

ImqBoolean setSendUserData(const char * *data* **= 0);**
Sets the **send user data** to the single item *data*. If *data* is not null, **send exit count** must be at least 1. This method returns TRUE if successful.

ImqBoolean setSendUserData(const size_t *count*, **const char** * *data* **[ ]);**
Sets the **send user data** to *data*. *count* must be no greater than the **send exit count**. This method returns TRUE if successful.

ImqBoolean setSendUserData(const size_t *count*, **const ImqString** * *data* **[ ]);**
Sets the **send user data** to *data*. *count* must be no greater than the **send exit count**. This method returns TRUE if successful.

ImqString sslCipherSpecification( ) const ;
Returns the SSL cipher specification.

ImqBoolean setSslCipherSpecification(const char * *name* **= 0);**
Sets the SSL cipher specification. This method returns TRUE if successful.

MQLONG sslClientAuthentication( ) const ;
Returns the SSL client authentication type.

ImqBoolean setSslClientAuthentication(const MQLONG *auth* **= MQSCA_REQUIRED);**
Sets the SSL client authentication type. This method returns TRUE if successful.

ImqString sslPeerName( ) const ;
Returns the SSL peer name.

ImqBoolean setSslPeerName(const char * *name* = 0);
Sets the SSL peer name. This method returns TRUE if successful.

ImqString transactionProgramName( ) const ;
Returns the **transaction program name**.

ImqBoolean setTransactionProgramName(const char * *name* = 0);
Sets the **transaction program name**. This method returns TRUE if successful.

MQLONG transportType( ) const ;
Returns the **transport type**.

ImqBoolean setTransportType(const MQLONG *type* = MQXPT_LU62);
Sets the **transport type**. This method returns TRUE if successful.

ImqString userId( ) const ;
Returns the **user id**.

ImqBoolean setUserId(const char * *id* = 0);
Sets the **user id**. This method returns TRUE if successful.

Reason codes

- MQRC_DATA_LENGTH_ERROR
- MQRC_ITEM_COUNT_ERROR
- MQRC_NULL_POINTER
- MQRC_SOURCE_BUFFER_ERROR

ImqCICSBridgeHeader C++ class

This class encapsulates specific features of the MQCIH data structure.

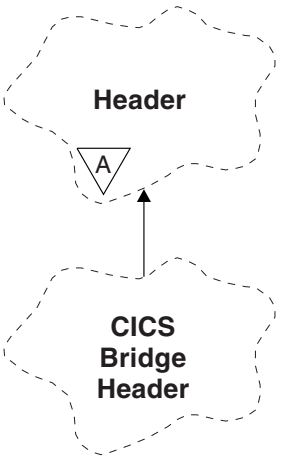


Figure 59. ImqCICSBridgeHeader class

Objects of this class are used by applications that send messages to the CICS bridge through WebSphere MQ for z/OS.

- "Return codes" on page 2673

Object attributes

ADS descriptor
Send/receive ADS descriptor. This is set using MQCADSD_NONE. The initial value is MQCADSD_NONE. The following additional values are possible:

- MQCADSD_NONE
- MQCADSD_SEND
- MQCADSD_RECV
- MQCADSD_MSGFORMAT

attention identifier
AID key. The field must be of length MQ_ATTENTION_ID_LENGTH.

authenticator
RACF password or passticket. The initial value contains blanks, of length MQ_AUTHENTICATOR_LENGTH.

bridge abend code
Bridge abend code, of length MQ_ABEND_CODE_LENGTH. The initial value is four blank characters. The value returned in this field is dependent on the return code. See Table 261 on page 2673 for more details.

bridge cancel code
Bridge abend transaction code. The field is reserved, must contain blanks, and be of length MQ_CANCEL_CODE_LENGTH.

bridge completion code
Completion code, which can contain either the WebSphere MQ completion code or the CICS EIBRESP value. The field has the initial value of MQCC_OK. The value returned in this field is dependent on the return code. See Table 261 on page 2673 for more details.

bridge error offset
Bridge error offset. The initial value is zero. This attribute is read-only.

bridge reason code
Reason code. This field can contain either the WebSphere MQ reason or the CICS EIBRESP2 value. The field has the initial value of MQRC_NONE. The value returned in this field is dependent on the return code. See Table 261 on page 2673 for more details.

bridge return code
Return code from the CICS bridge. The initial value is MQCRC_OK.

conversational task
Whether the task can be conversational. The initial value is MQCCT_NO. The following additional values are possible:

- MQCCT_YES
- MQCCT_NO

cursor position
Cursor position. The initial value is zero.

facility keep time
CICS bridge facility release time.

facility like
Terminal emulated attribute. The field must be of length MQ_FACILITY_LIKE_LENGTH.

facility token
BVT token value. The field must be of length MQ_FACILITY_LENGTH. The initial value is MQCFAC_NONE.

function
Function, which can contain either the WebSphere MQ call name or the CICS EIBFN function. The field has the initial value of MQCFUNC_NONE, with length MQ_FUNCTION_LENGTH. The value returned in this field is dependent on the return code. See Table 261 on page 2673 for more details.

The following additional values are possible when **function** contains a WebSphere MQ call name:

- MQCFUNC_MQCONN
- MQCFUNC_MQGET
- MQCFUNC_MQINQ
- MQCFUNC_NONE
- MQCFUNC_MQOPEN
- MQCFUNC_PUT
- MQCFUNC_MQPUT1

get wait interval
Wait interval for an MQGET call issued by the CICS bridge task. The initial value is MQCGWI_DEFAULT. The field applies only when **uow control** has the value MQCUOWC_FIRST. The following additional values are possible:

- MQCGWI_DEFAULT
- MQWI_UNLIMITED

link type
Link type. The initial value is MQCLT_PROGRAM. The following additional values are possible:

- MQCLT_PROGRAM
- MQCLT_TRANSACTION

next transaction identifier
ID of the next transaction to attach. The field must be of length MQ_TRANSACTION_ID_LENGTH.

output data length
COMMAREA data length. The initial value is MQCODL_AS_INPUT.

reply-to format
Format name of the reply message. The initial value is MQFMT_NONE with length MQ_FORMAT_LENGTH.

start code
Transaction start code. The field must be of length MQ_START_CODE_LENGTH. The initial value is MQCSC_NONE. The following additional values are possible:

- MQCSC_START
- MQCSC_STARTDATA
- MQCSC_TERMINPUT
- MQCSC_NONE

task end status
Task end status. The initial value is MQCTES_NOSYNC. The following additional values are possible:

- MQCTES_COMMIT
- MQCTES_BACKOUT
- MQCTES_ENDTASK

- MQCTES_NOSYNC

transaction identifier
ID of the transaction to attach. The initial value must contain blanks, and must be of length MQ_TRANSACTION_ID_LENGTH. The field applies only when **uow control** has the value MQCUOWC_FIRST or MQCUOWC_ONLY.

UOW control
UOW control. The initial value is MQCUOWC_ONLY. The following additional values are possible:

- MQCUOWC_FIRST
- MQCUOWC_MIDDLE
- MQCUOWC_LAST
- MQCUOWC_ONLY
- MQCUOWC_COMMIT
- MQCUOWC_BACKOUT
- MQCUOWC_CONTINUE

version
The MQCIH version number. The initial value is MQCIH_VERSION_2. The only other supported value is MQCIH_VERSION_1.

Constructors

ImqCICSBridgeHeader( );
The default constructor.

ImqCICSBridgeHeader(const ImqCICSBridgeHeader & *header* **);**
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg* **);**
Inserts an MQCIH data structure into the message buffer at the beginning, moving existing message data further along, and sets the message format to MQFMT_CICS.

See the parent class method description for more details.

virtual ImqBoolean pasteIn(ImqMessage & *msg* **);**
Reads an MQCIH data structure from the message buffer. To be successful, the encoding of the *msg* object must be MQENC_NATIVE. Retrieve messages with MQGMO_CONVERT to MQENC_NATIVE. To be successful, the ImqMessage format must be MQFMT_CICS.

See the parent class method description for more details.

Object methods (public)

void operator = (const ImqCICSBridgeHeader & *header* **);**
Copies instance data from the *header*, replacing the existing instance data.

MQLONG ADSDescriptor( ) const;
Returns a copy of the **ADS descriptor**.

void setADSDescriptor(const MQLONG *descriptor* **= MQCADSD_NONE);**
Sets the **ADS descriptor**.

ImqString attentionIdentifier( ) const;
Returns a copy of the **attention identifier**, padded with trailing blanks to length MQ_ATTENTION_ID_LENGTH.

void setAttentionIdentifier(const char * *data* = 0);
Sets the **attention identifier**, padded with trailing blanks to length MQ_ATTENTION_ID_LENGTH. If no *data* is supplied, resets **attention identifier** to the initial value.

ImqString authenticator( ) const;
Returns a copy of the **authenticator**, padded with trailing blanks to length MQ_AUTHENTICATOR_LENGTH.

void setAuthenticator(const char * *data* = 0);
Sets the **authenticator**, padded with trailing blanks to length MQ_AUTHENTICATOR_LENGTH. If no *data* is supplied, resets **authenticator** to the initial value.

ImqString bridgeAbendCode( ) const;
Returns a copy of the **bridge abend code**, padded with trailing blanks to length MQ_ABEND_CODE_LENGTH.

ImqString bridgeCancelCode( ) const;
Returns a copy of the **bridge cancel code**, padded with trailing blanks to length MQ_CANCEL_CODE_LENGTH.

void setBridgeCancelCode(const char * *data* = 0);
Sets the **bridge cancel code**, padded with trailing blanks to length MQ_CANCEL_CODE_LENGTH. If no *data* is supplied, resets the **bridge cancel code** to the initial value.

MQLONG bridgeCompletionCode( ) const;
Returns a copy of the **bridge completion code**.

MQLONG bridgeErrorOffset( ) const ;
Returns a copy of the **bridge error offset**.

MQLONG bridgeReasonCode( ) const;
Returns a copy of the **bridge reason code**.

MQLONG bridgeReturnCode( ) const;
Returns the **bridge return code**.

MQLONG conversationalTask( ) const;
Returns a copy of the **conversational task**.

void setConversationalTask(const MQLONG *task* = MQCCT_NO);
Sets the **conversational task**.

MQLONG cursorPosition( ) const ;
Returns a copy of the **cursor position**.

void setCursorPosition(const MQLONG *position* = 0);
Sets the **cursor position**.

MQLONG facilityKeepTime( ) const;
Returns a copy of the **facility keep time**.

void setFacilityKeepTime(const MQLONG *time* = 0);
Sets the **facility keep time**.

ImqString facilityLike( ) const;
Returns a copy of the **facility like**, padded with trailing blanks to length MQ_FACILITY_LIKE_LENGTH.

void setFacilityLike(const char * *name* = 0);
Sets the **facility like**, padded with trailing blanks to length MQ_FACILITY_LIKE_LENGTH. If no *name* is supplied, resets **facility like** the initial value.

ImqBinary facilityToken( ) const;
Returns a copy of the **facility token**.

ImqBoolean setFacilityToken(const ImqBinary & *token* **);**
Sets the **facility token**. The **data length** of *token* must be either zero or MQ_FACILITY_LENGTH. It returns TRUE if successful.

void setFacilityToken(const MQBYTE8 *token* **= 0);**
Sets the **facility token**. *token* can be zero, which is the same as specifying MQCFAC_NONE. If *token* is nonzero it must address MQ_FACILITY_LENGTH bytes of binary data. When using predefined values such as MQCFAC_NONE, you might need to make a cast to ensure a signature match. For example, (MQBYTE *)MQCFAC_NONE.

ImqString function( ) const;
Returns a copy of the **function**, padded with trailing blanks to length MQ_FUNCTION_LENGTH.

MQLONG getWaitInterval( ) const;
Returns a copy of the **get wait interval**.

void setGetWaitInterval(const MQLONG *interval* **= MQCGWI_DEFA**
Sets the **get wait interval**.

MQLONG linkType( ) const;
Returns a copy of the **link type**.

void setLinkType(const MQLONG *type* **= MQCLT_PROGRAM);**
Sets the **link type**.

ImqString nextTransactionIdentifier( ) const ;
Returns a copy of the **next transaction identifier** data, padded with trailing blanks to length MQ_TRANSACTION_ID_LENGTH.

MQLONG outputDataLength( ) const;
Returns a copy of the **output data length**.

void setOutputDataLength(const MQLONG *length* **= MQCODL_AS_INPUT);**
Sets the **output data length**.

ImqString replyToFormat( ) const;
Returns a copy of the **reply-to format** name, padded with trailing blanks to length MQ_FORMAT_LENGTH.

void setReplyToFormat(const char * *name* **= 0);**
Sets the **reply-to format**, padded with trailing blanks to length MQ_FORMAT_LENGTH. If no *name* is supplied, resets **reply-to format** to the initial value.

ImqString startCode( ) const;
Returns a copy of the **start code**, padded with trailing blanks to length MQ_START_CODE_LENGTH.

void setStartCode(const char * *data* **= 0);**
Sets the **start code** data, padded with trailing blanks to length MQ_START_CODE_LENGTH. If no *data* is supplied, resets **start code** to the initial value.

MQLONG taskEndStatus( ) const;
Returns a copy of the **task end status**.

ImqString transactionIdentifier( ) const;
Returns a copy of the **transaction identifier** data, padded with trailing blanks to the length MQ_TRANSACTION_ID_LENGTH.

void setTransactionIdentifier(const char * *data* = 0);
Sets the **transaction identifier**, padded with trailing blanks to length MQ_TRANSACTION_ID_LENGTH. If no *data* is supplied, resets **transaction identifier** to the initial value.

MQLONG UOWControl( ) const;
Returns a copy of the **UOW control**.

void setUOWControl(const MQLONG *control* = MQCUOWC_ONLY);
Sets the **UOW control**.

MQLONG version( ) const;
Returns the **version** number.

ImqBoolean setVersion(const MQLONG *version* = MQCIH_VERSION_2);
Sets the **version** number. It returns TRUE if successful.

Object data (protected)

MQLONG *olVersion*
The maximum MQCIH version number that can be accommodated in the storage allocated for *opcih*.

PMQCIH *opcih*
The address of an MQCIH data structure. The amount of storage allocated is indicated by *olVersion*.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR
- MQRC_WRONG_VERSION

Return codes

Table 261. ImqCICSBridgeHeader class return codes

Return Code	Function	CompCode	Reason	Abend Code
MQCRC_OK				
MQCRC_BRIDGE_ERROR			MQFB_CICS	
MQCRC_MQ_API_ERROR	WebSphere MQ call name	WebSphere MQ CompCode	WebSphere MQ Reason	
MQCRC_BRIDGE_TIMEOUT	WebSphere MQ call name	WebSphere MQ CompCode	WebSphere MQ Reason	
MQCRC_CICS_EXEC_ERROR	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	
MQCRC_SECURITY_ERROR	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	
MQCRC_PROGRAM_NOT_AVAILABLE	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	
MQCRC_TRANSID_NOT_AVAILABLE	CICS EIBFN	CICS EIBRESP	CICS EIBRESP2	
MQCRC_BRIDGE_ABEND				CICS ABCODE
MQCRC_APPLICATION_ABEND				CICS ABCODE

ImqDeadLetterHeader C++ class

This class encapsulates features of the MQDLH data structure.

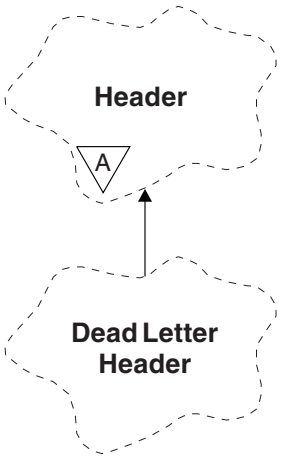


Figure 60. ImqDeadLetterHeader class

Objects of this class are typically used by an application that encounters an message that cannot be processed. A new message comprising a dead-letter header and the message content is placed on the dead-letter queue, and the message is discarded.

Object attributes

dead-letter reason code
The reason the message arrived on the dead-letter queue. The initial value is MQRC_NONE.

destination queue manager name
The name of the original destination queue manager. The name is a string of length MQ_Q_MGR_NAME_LENGTH. Its initial value is null.

destination queue name
The name of the original destination queue. The name is a string of length MQ_Q_NAME_LENGTH. Its initial value is null.

put application name
The name of the application that put the message on the dead-letter queue. The name is a string of length MQ_PUT_APPL_NAME_LENGTH. Its initial value is null.

put application type
The type of application that put the message on the dead-letter queue. The initial value is zero.

put date
The date when the message was put on the dead-letter queue. The date is a string of length MQ_PUT_DATE_LENGTH. Its initial value is a null string.

put time
The time when the message was put on the dead-letter queue. The time is a string of length MQ_PUT_TIME_LENGTH. Its initial value is a null string.

Constructors

ImqDeadLetterHeader( );
The default constructor.

ImqDeadLetterHeader(const ImqDeadLetterHeader & *header*);
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg*);
Inserts an MQDLH data structure into the message buffer at the beginning, moving existing message data further along. Sets the *msg* **format** to MQFMT_DEAD_LETTER_HEADER.

See the ImqHeader class method description on page "ImqHeader C++ class" on page 2682 for further details.

virtual ImqBoolean pasteIn(ImqMessage & *msg*);
Reads an MQDLH data structure from the message buffer.

To be successful, the ImqMessage **format** must be MQFMT_DEAD_LETTER_HEADER.

See the ImqHeader class method description on page "ImqHeader C++ class" on page 2682 for further details.

Object methods (public)

void operator = (const ImqDeadLetterHeader & *header*);
Copies instance data is copied from *header*, replacing the existing instance data.

MQLONG deadLetterReasonCode( ) const ;
Returns the **dead-letter reason code**.

void setDeadLetterReasonCode(const MQLONG *reason*);
Sets the **dead-letter reason code**.

ImqString destinationQueueManagerName( ) const ;
Returns the **destination queue manager name**, stripped of any trailing blanks.

void setDestinationQueueManagerName(const char * *name*);
Sets the **destination queue manager name**. Truncates data longer than MQ_Q_MGR_NAME_LENGTH (48 characters).

ImqString destinationQueueName( ) const ;
Returns a copy of the **destination queue name**, stripped of any trailing blanks.

void setDestinationQueueName(const char * *name*);
Sets the **destination queue name**. Truncates data longer than MQ_Q_NAME_LENGTH (48 characters).

ImqString putApplicationName( ) const ;
Returns a copy of the **put application name**, stripped of any trailing blanks.

void setPutApplicationName(const char * *name* = 0);
Sets the **put application name**. Truncates data longer than MQ_PUT_APPL_NAME_LENGTH (28 characters).

MQLONG putApplicationType( ) const ;
Returns the **put application type**.

void setPutApplicationType(const MQLONG *type* = MQAT_NO_CONTEXT);
Sets the **put application type**.

ImqString putDate( ) const ;
Returns a copy of the **put date**, stripped of any trailing blanks.

void setPutDate(const char * *date* = 0);
Sets the **put date**. Truncates data longer than MQ_PUT_DATE_LENGTH (8 characters).

ImqString putTime( ) const ;
Returns a copy of the **put time**, stripped of any trailing blanks.

void setPutTime(const char * *time* = 0);
Sets the **put time**. Truncates data longer than MQ_PUT_TIME_LENGTH (8 characters).

Object data (protected)

MQDLH *omqdlh*
The MQDLH data structure.

Reason codes

- MQRC_INCONSISTENT_FORMAT
- MQRC_STRUC_ID_ERROR
- MQRC_ENCODING_ERROR

ImqDistributionList C++ class

This class encapsulates a dynamic distribution list that references one or more queues for the purpose of sending a message or messages to multiple destinations.

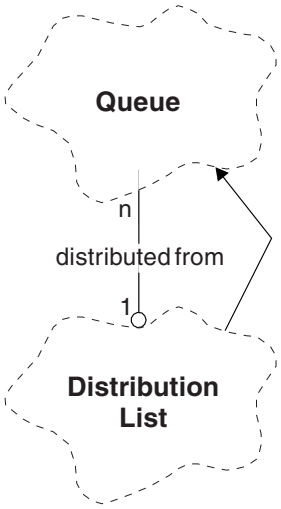


Figure 61. ImqDistributionList class

- "Object attributes"
- "Constructors" on page 2677
- "Object methods (public)" on page 2677
- "Object methods (protected)" on page 2677

Object attributes

first distributed queue
The first of one or more objects of class , in no particular order, in which the **distribution list reference** addresses this object.

Initially there are no such objects. To open an ImqDistributionList successfully, there must be at least one such object.

Note: When an ImqDistributionList object is opened, any open objects that reference it are automatically closed.

Constructors

ImqDistributionList( );
The default constructor.

ImqDistributionList(const ImqDistributionList & *list* **);**
The copy constructor.

Object methods (public)

void operator = (const ImqDistributionList & *list* **);**
All objects that reference **this** object are dereferenced before copying. No objects will reference **this** object after the invocation of this method.

*** firstDistributedQueue( ) const ;**
Returns the **first distributed queue**.

Object methods (protected)

void setFirstDistributedQueue(* *queue* **= 0);**
Sets the **first distributed queue**.

ImqError C++ class

This abstract class provides information on errors associated with an object.

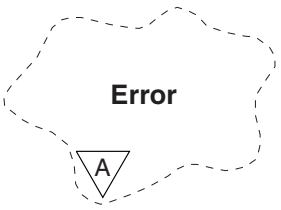


Figure 62. ImqError class

Object attributes

completion code
The most recent completion code. The initial value is zero. The following additional values are possible:

- MQCC_OK
- MQCC_WARNING
- MQCC_FAILED

reason code
The most recent reason code. The initial value is zero.

Constructors

ImqError( );
The default constructor.

ImqError(const ImqError & *error* **);**
The copy constructor.

Object methods (public)

void operator = (const ImqError & *error* **);**
Copies instance data from *error*, replacing the existing instance data.

void clearErrorCodes( );
Sets the **completion code** and **reason code** both to zero.

MQLONG completionCode( ) const ;
Returns the **completion code**.

MQLONG reasonCode( ) const ;
Returns the **reason code**.

Object methods (protected)

ImqBoolean checkReadPointer(const void * *pointer*, **const size_t** *length* **);**
Verifies that the combination of pointer and length is valid for read-only access, and returns TRUE if successful.

ImqBoolean checkWritePointer(const void * *pointer*, **const size_t** *length* **);**
Verifies that the combination of pointer and length is valid for read-write access, and returns TRUE if successful.

void setCompletionCode(const MQLONG *code* **= 0);**
Sets the **completion code**.

void setReasonCode(const MQLONG *code* **= 0);**
Sets the **reason code**.

Reason codes

- MQRC_BUFFER_ERROR

ImqGetMessageOptions C++ class

This class encapsulates the MQGMO data structure

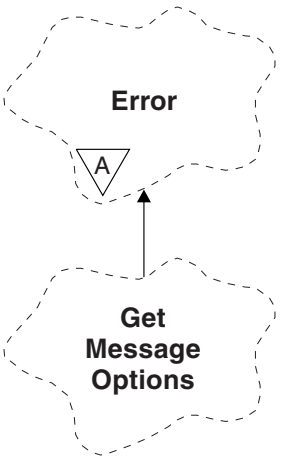


Figure 63. ImqGetMessageOptions class

Object attributes

group status
Status of a message for a group of messages. The initial value is MQGS_NOT_IN_GROUP. The following additional values are possible:

- MQGS_MSG_IN_GROUP
- MQGS_LAST_MSG_IN_GROUP

match options
Options for selecting incoming messages. The initial value is MQMO_MATCH_MSG_ID | MQMO_MATCH_CORREL_ID. The following additional values are possible:

- MQMO_GROUP_ID
- MQMO_MATCH_MSG_SEQ_NUMBER
- MQMO_MATCH_OFFSET
- MQMO_MSG_TOKEN
- MQMO_NONE

message token
Message token. A binary value (MQBYTE16) of length MQ_MSG_TOKEN_LENGTH. The initial value is MQMTOK_NONE.

options
Options applicable to a message. The initial value is MQGMO_NO_WAIT. The following additional values are possible:

- MQGMO_WAIT
- MQGMO_SYNCPOINT
- MQGMO_SYNCPOINT_IF_PERSISTENT
- MQGMO_NO_SYNCPOINT

- MQGMO_MARK_SKIP_BACKOUT
- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQGMO_MSG_UNDER_CURSOR
- MQGMO_LOCK
- MQGMO_UNLOCK
- MQGMO_ACCEPT_TRUNCATED_MSG
- MQGMO_SET_SIGNAL
- MQGMO_FAIL_IF_QUIESCING
- MQGMO_CONVERT
- MQGMO_LOGICAL_ORDER
- MQGMO_COMPLETE_MSG
- MQGMO_ALL_MSGS_AVAILABLE
- MQGMO_ALL_SEGMENTS_AVAILABLE
- MQGMO_NONE

resolved queue name
Resolved queue name. This attribute is read-only. Names are never longer than 48 characters and can be padded to that length with nulls. The initial value is a null string.

returned length
Returned length. The initial value is MQRL_UNDEFINED. This attribute is read-only.

segmentation
The ability to segment a message. The initial value is MQSEG_INHIBITED. The additional value, MQSEG_ALLOWED, is possible.

segment status
The segmentation status of a message. The initial value is MQSS_NOT_A_SEGMENT. The following additional values are possible:

- MQSS_SEGMENT
- MQSS_LAST_SEGMENT

syncpoint participation
TRUE when messages are retrieved under syncpoint control.

wait interval
The length of time that the class **get** method pauses while waiting for a suitable message to arrive, if one is not already available. The initial value is zero, which effects an indefinite wait. The additional value, MQWI_UNLIMITED, is possible. This attribute is ignored unless the **options** include MQGMO_WAIT.

Constructors

ImqGetMessageOptions( );
The default constructor.

ImqGetMessageOptions(const ImqGetMessageOptions & *gmo* **);**
The copy constructor.

Object methods (public)

void operator = (const ImqGetMessageOptions & *gmo* **);**
Copies instance data from *gmo*, replacing the existing instance data.

MQCHAR groupStatus( ) const ;
Returns the **group status**.

void setGroupStatus(const MQCHAR *status* **);**
Sets the **group status**.

MQLONG matchOptions( ) const ;
Returns the **match options**.

void setMatchOptions(const MQLONG *options* **);**
Sets the **match options**.

ImqBinary messageToken( ) const;
Returns the **message token**.

ImqBoolean setMessageToken(const ImqBinary & *token* **);**
Sets the **message token**. The **data length** of *token* must be either zero or MQ_MSG_TOKEN_LENGTH. This method returns TRUE if successful.

void setMessageToken(const MQBYTE16 *token* **= 0);**
Sets the **message token**. *token* can be zero, which is the same as specifying MQMTOK_NONE. If *token* is nonzero, then it must address MQ_MSG_TOKEN_LENGTH bytes of binary data.

When using predefined values, such as MQMTOK_NONE, you might not need to make a cast to ensure a signature match, for example (MQBYTE *)MQMTOK_NONE.

MQLONG options( ) const ;
Returns the **options**.

void setOptions(const MQLONG *options* **);**
Sets the **options**, including the **syncpoint participation** value.

ImqString resolvedQueueName( ) const ;
Returns a copy of the **resolved queue name**.

MQLONG returnedLength( ) const;
Returns the **returned length**.

MQCHAR segmentation( ) const ;
Returns the **segmentation**.

void setSegmentation(const MQCHAR *value* **);**
Sets the **segmentation**.

MQCHAR segmentStatus( ) const ;
Returns the **segment status**.

void setSegmentStatus(const MQCHAR *status* **);**
Sets the **segment status**.

ImqBoolean syncPointParticipation( ) const ;
Returns the **syncpoint participation** value, which is TRUE if the **options** include either MQGMO_SYNCPOINT or MQGMO_SYNCPOINT_IF_PERSISTENT.

void setSyncPointParticipation(const ImqBoolean *sync* **);**
Sets the **syncpoint participation** value. If *sync* is TRUE, alters the **options** to include MQGMO_SYNCPOINT, and to exclude both MQGMO_NO_SYNCPOINT and MQGMO_SYNCPOINT_IF_PERSISTENT. If *sync* is FALSE, alters the **options** to include MQGMO_NO_SYNCPOINT, and to exclude both MQGMO_SYNCPOINT and MQGMO_SYNCPOINT_IF_PERSISTENT.

MQLONG waitInterval( ) const ;
Returns the **wait interval**.

void setWaitInterval(const MQLONG *interval* **);**
Sets the **wait interval**.

Object methods (protected)

static void setVersionSupported(const MQLONG);
Sets the **MQGMO** version. Defaults to **MQGMO_VERSION_3**.

Object data (protected)

MQGMO *omqgmo*
An MQGMO Version 2 data structure. Access MQGMO fields supported for MQGMO_VERSION_2 only.

PMQGMO *opgmo*
The address of an MQGMO data structure. The version number for this address is indicated in *olVersion*. Inspect the version number before accessing MQGMO fields, to ensure that they are present.

MQLONG *olVersion*
The version number of the MQGMO data structure addressed by *opgmo*.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR

ImqHeader C++ class

This abstract class encapsulates common features of the MQDLH data structure.

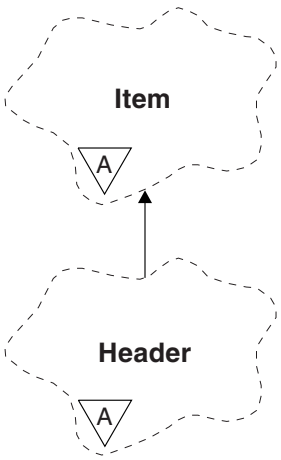


Figure 64. ImqHeader class

- "Object attributes"
- "Constructors" on page 2683
- "Object methods (public)" on page 2683

Object attributes

character set
The original coded character set identifier. Initially MQCCSI_Q_MGR.

encoding
The original encoding. Initially MQENC_NATIVE.

format
The original format. Initially MQFMT_NONE.

header flags
The initial values are:

- Zero for objects of the ImqDeadLetterHeader class
- MQIIH_NONE for objects of the ImqIMSBridgeHeader class
- MQRMHF_LAST for objects of the ImqReferenceHeader class
- MQCIH_NONE for objects of the ImqCICSBridgeHeader class
- MQWIH_NONE for objects of the ImqWorkHeader class

Constructors

ImqHeader( );
The default constructor.

ImqHeader(const ImqHeader & *header* **);**
The copy constructor.

Object methods (public)

void operator = (const ImqHeader & *header* **);**
Copies instance data from *header*, replacing the existing instance data.

virtual MQLONG characterSet( ) const ;
Returns the **character set**.

virtual void setCharacterSet(const MQLONG *ccsid* **= MQCCSI_Q_MGR);**
Sets the **character set**.

virtual MQLONG encoding( ) const ;
Returns the **encoding**.

virtual void setEncoding(const MQLONG *encoding* **= MQENC_NATIVE);**
Sets the **encoding**.

virtual ImqString format( ) const ;
Returns a copy of the **format**, including trailing blanks.

virtual void setFormat(const char * *name* **= 0);**
Sets the **format**, padded to 8 characters with trailing blanks.

virtual MQLONG headerFlags( ) const ;
Returns the **header flags**.

virtual void setHeaderFlags(const MQLONG *flags* **= 0);**
Sets the **header flags**.

ImqIMSBridgeHeader C++ class

This class encapsulates features of the MQIIH data structure.

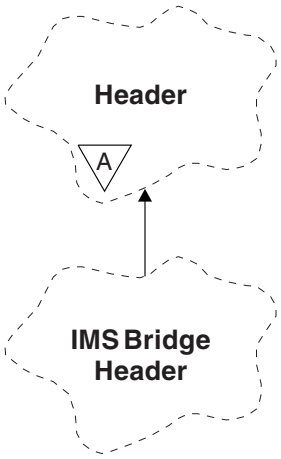


Figure 65. ImqIMSBridgeHeader class

Objects of this class are used by applications that send messages to the IMS bridge through WebSphere MQ for z/OS.

Note: The ImqHeader **character set** and **encoding** must have default values and must not be set to any other values.

Object attributes

authenticator
RACF password or passticket, of length MQ_AUTHENTICATOR_LENGTH. The initial value is MQIAUT_NONE.

commit mode
Commit mode. See the *OTMA User's Guide* for more information about IMS commit modes. The initial value is MQICM_COMMIT_THEN_SEND. The additional value, MQICM_SEND_THEN_COMMIT, is possible.

logical terminal override
Logical terminal override, of length MQ_LTERM_OVERRIDE_LENGTH. The initial value is a null string.

message format services map name
MFS map name, of length MQ_MFS_MAP_NAME_LENGTH. The initial value is a null string.

reply-to format
Format of any reply, of length MQ_FORMAT_LENGTH. The initial value is MQFMT_NONE.

security scope
Scope of IMS security processing. The initial value is MQISS_CHECK. The additional value, MQISS_FULL, is possible.

transaction instance id
Transaction instance identity, a binary (MQBYTE16) value of length MQ_TRAN_INSTANCE_ID_LENGTH. The initial value is MQITII_NONE.

transaction state
State of the IMS conversation. The initial value is MQITS_NOT_IN_CONVERSATION. The additional value, MQITS_IN_CONVERSATION, is possible.

Constructors

ImqIMSBridgeHeader( );
The default constructor.

ImqIMSBridgeHeader(const ImqIMSBridgeHeader & *header* **);**
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg* **);**
Inserts an MQIIH data structure into the message buffer at the beginning, moving existing message data further along. Sets the *msg* **format** to MQFMT_IMS.

See the parent class method description for further details.

virtual ImqBoolean pasteIn(ImqMessage & *msg* **);**
Reads an MQIIH data structure from the message buffer.

To be successful, the **encoding** of the *msg* object must be MQENC_NATIVE. Retrieve messages with MQGMO_CONVERT to MQENC_NATIVE.

To be successful, the ImqMessage **format** must be MQFMT_IMS.

See the parent class method description for further details.

Object methods (public)

void operator = (const ImqIMSBridgeHeader & *header* **);**
Copies instance data from *header*, replacing the existing instance data.

ImqString authenticator( ) const ;
Returns a copy of the **authenticator**, padded with trailing blanks to length MQ_AUTHENTICATOR_LENGTH.

void setAuthenticator(const char * *name* **);**
Sets the **authenticator**.

MQCHAR commitMode( ) const ;
Returns the **commit mode**.

void setCommitMode(const MQCHAR *mode* **);**
Sets the **commit mode**.

ImqString logicalTerminalOverride( ) const ;
Returns a copy of the **logical terminal override**.

void setLogicalTerminalOverride(const char * *override* **);**
Sets the **logical terminal override**.

ImqString messageFormatServicesMapName( ) const ;
Returns a copy of the **message format services map name**.

void setMessageFormatServicesMapName(const char * *name* **);**
Sets the **message format services map name**.

ImqString replyToFormat( ) const ;
Returns a copy of the **reply-to format**, padded with trailing blanks to length MQ_FORMAT_LENGTH.

void setReplyToFormat(const char * *format*);
Sets the **reply-to format**, padded with trailing blanks to length MQ_FORMAT_LENGTH.

MQCHAR securityScope( ) const ;
Returns the **security scope**.

void setSecurityScope(const MQCHAR *scope*);
Sets the **security scope**.

ImqBinary transactionInstanceId( ) const ;
Returns a copy of the **transaction instance id**.

ImqBoolean setTransactionInstanceId(const ImqBinary & *id*);
Sets the **transaction instance id**. The **data length** of *token* must be either zero or MQ_TRAN_INSTANCE_ID_LENGTH. This method returns TRUE if successful.

void setTransactionInstanceId(const MQBYTE16 *id* = 0);
Sets the **transaction instance id**. *id* can be zero, which is the same as specifying MQITII_NONE. If *id* is nonzero, it must address MQ_TRAN_INSTANCE_ID_LENGTH bytes of binary data. When using predefined values such as MQITII_NONE, you might need to make a cast to ensure a signature match, for example (MQBYTE *)MQITII_NONE.

MQCHAR transactionState( ) const ;
Returns the **transaction state**.

void setTransactionState(const MQCHAR *state*);
Sets the **transaction state**.

Object data (protected)

MQIIH *omqiih*
The MQIIH data structure.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR
- MQRC_INCONSISTENT_FORMAT
- MQRC_ENCODING_ERROR
- MQRC_STRUC_ID_ERROR

ImqItem C++ class

This abstract class represents an item, perhaps one of several, within a message.

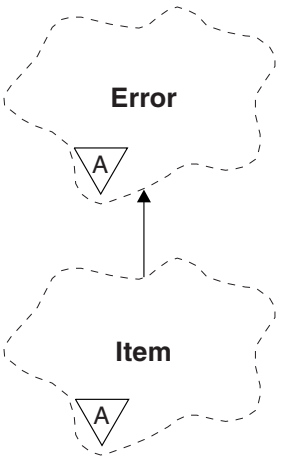


Figure 66. ImqItem class

Items are concatenated together in a message buffer. Each specialization is associated with a particular data structure that begins with a structure id.

Polymorphic methods in this abstract class allow items to be copied to and from messages. The ImqMessage class **readItem** and **writeItem** methods provide another style of invoking these polymorphic methods that is more natural for application programs.

Object attributes

structure id
A string of four characters at the beginning of the data structure. This attribute is read-only. Consider this attribute for derived classes. It is not included automatically.

Constructors

ImqItem( );
The default constructor.

ImqItem(const ImqItem & *item* **);**
The copy constructor.

Class methods (public)

static ImqBoolean structureIdIs(const char * *structure-id-to-test*, **const ImqMessage &** *msg* **);**
Returns TRUE if the **structure id** of the next ImqItem in the incoming *msg* is the same as *structure-id-to-test*. The next item is identified as that part of the message buffer currently addressed by the ImqCache **data pointer**. This method relies on the **structure id** and therefore is not guaranteed to work for all ImqItem derived classes.

Object methods (public)

void operator = (const ImqItem & *item* **);**
Copies instance data from *item,* replacing the existing instance data.

virtual ImqBoolean copyOut(ImqMessage & *msg*) = 0 ;

Writes this object as the next item in an outgoing message buffer, appending it to any existing items. If the write operation is successful, increases the ImqCache **data length**. This method returns TRUE if successful.

Override this method to work with a specific subclass.

virtual ImqBoolean pasteIn(ImqMessage & *msg*) = 0 ;

Reads this object *destructively* from the incoming message buffer. The read is destructive in that the ImqCache **data pointer** is moved on. However, the buffer content remains the same, so data can be re-read by resetting the ImqCache **data pointer**.

The (sub)class of this object must be consistent with the **structure id** found next in the message buffer of the *msg* object.

The **encoding** of the *msg* object should be MQENC_NATIVE. It is recommended that messages be retrieved with the ImqMessage **encoding** set to MQENC_NATIVE, and with the ImqGetMessageOptions **options** including MQGMO_CONVERT.

If the read operation is successful, the ImqCache **data length** is reduced. This method returns TRUE if successful.

Override this method to work with a specific subclass.

Reason codes

- MQRC_ENCODING_ERROR
- MQRC_STRUC_ID_ERROR
- MQRC_INCONSISTENT_FORMAT
- MQRC_INSUFFICIENT_BUFFER
- MQRC_INSUFFICIENT_DATA

ImqMessage C++ class

This class encapsulates an MQMD data structure and also handles the construction and reconstruction of message data.

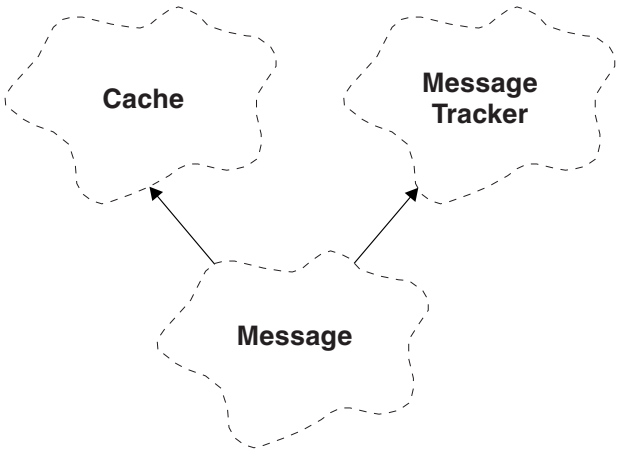


Figure 67. ImqMessage class

- "Object attributes" on page 2689
- "Constructors" on page 2692
- "Object methods (public)" on page 2692
- "Object methods (protected)" on page 2695

- "Object data (protected)" on page 2695

Object attributes

application id data
Identity information associated with a message. The initial value is a null string.

application origin data
Origin information associated with a message. The initial value is a null string.

backout count
The number of times that a message has been tentatively retrieved and subsequently backed out. The initial value is zero. This attribute is read-only.

character set
Coded Character Set Id. The initial value is MQCCSI_Q_MGR. The following additional values are possible:

- MQCCSI_INHERIT
- MQCCSI_EMBEDDED

You can also use a Coded Character Set Id of your choice. For information about this, see "Code page conversion" on page 2241.

encoding
The machine encoding of the message data. The initial value is MQENC_NATIVE.

expiry A time-dependent quantity that controls how long WebSphere MQ retains an unretrieved message before discarding it. The initial value is MQEI_UNLIMITED.

format
The name of the format (template) that describes the layout of data in the buffer. Names longer than eight characters are truncated to eight characters. Names are always padded with blanks to eight characters. The initial constant value is MQFMT_NONE. The following additional constants are possible:

- MQFMT_ADMIN
- MQFMT_CICS
- MQFMT_COMMAND_1
- MQFMT_COMMAND_2
- MQFMT_DEAD_LETTER_HEADER
- MQFMT_DIST_HEADER
- MQFMT_EVENT
- MQFMT_IMS
- MQFMT_IMS_VAR_STRING
- MQFMT_MD_EXTENSION
- MQFMT_PCF
- MQFMT_REF_MSG_HEADER
- MQFMT_RF_HEADER
- MQFMT_STRING
- MQFMT_TRIGGER
- MQFMT_WORK_INFO_HEADER
- MQFMT_XMIT_Q_HEADER

You can also use an application-specific string of your choice. For more information about this, see the "Format (MQCHAR8)" on page 1721 field of the message descriptor (MQMD).

message flags
Segmentation control information. The initial value is MQMF_SEGMENTATION_INHIBITED. The following additional values are possible:

- MQMF_SEGMENTATION_ALLOWED
- MQMF_MSG_IN_GROUP
- MQMF_LAST_MSG_IN_GROUP
- MQMF_SEGMENT
- MQMF_LAST_SEGMENT
- MQMF_NONE

message type
The broad categorization of a message. The initial value is MQMT_DATAGRAM. The following additional values are possible:

- MQMT_SYSTEM_FIRST
- MQMT_SYSTEM_LAST
- MQMT_DATAGRAM
- MQMT_REQUEST
- MQMT_REPLY
- MQMT_REPORT
- MQMT_APPL_FIRST
- MQMT_APPL_LAST

You can also use an application-specific value of your choice. For more information about this, see the "MsgType (MQLONG)" on page 1732 field of the message descriptor (MQMD).

offset Offset information. The initial value is zero.

original length
The original length of a segmented message. The initial value is MQOL_UNDEFINED.

persistence
Indicates that the message is important and must at all times be backed up using persistent storage. This option implies a performance penalty. The initial value is MQPER_PERSISTENCE_AS_Q_DEF. The following additional values are possible:

- MQPER_PERSISTENT
- MQPER_NOT_PERSISTENT

priority
The relative priority for transmission and delivery. Messages of the same priority are usually delivered in the same sequence as they were supplied (although there are several criteria that must be satisfied to guarantee this). The initial value is MQPRI_PRIORITY_AS_Q_DEF.

property validation
Specifies whether validation of properties should take place when a property of the message is set. The initial value is **MQCMHO_DEFAULT_VALIDATION**. The following additional values are possible:

- MQCMHO_VALIDATE
- MQCMHO_NO_VALIDATION

The following methods act on **property validation**:

MQLONG propertyValidation( ) const ;
Returns the **property validation** option.

void setPropertyValidation(const MQLONG *option* **);**
Sets the **property validation** option.

put application name
The name of the application that put a message. The initial value is a null string.

put application type
The type of application that put a message. The initial value is MQAT_NO_CONTEXT. The following additional values are possible:

- MQAT_AIX
- MQAT_CICS
- MQAT_CICS_BRIDGE
- MQAT_DOS
- MQAT_IMS
- MQAT_IMS_BRIDGE
- MQAT_MVS
- MQAT_NOTES_AGENT
- MQAT_OS2
- MQAT_OS390
- MQAT_OS400
- MQAT_QMGR
- MQAT_UNIX
- MQAT_WINDOWS
- MQAT_WINDOWS_NT
- MQAT_XCF
- MQAT_DEFAULT
- MQAT_UNKNOWN
- MQAT_USER_FIRST
- MQAT_USER_LAST

You can also use an application-specific string of your choice. For more information about this, see the "PutApplType (MQLONG)" on page 1738 field of the message descriptor (MQMD).

put date
The date on which a message was put. The initial value is a null string.

put time
The time at which a message was put. The initial value is a null string.

reply-to queue manager name
The name of the queue manager to which any reply should be sent. The initial value is a null string.

reply-to queue name
The name of the queue to which any reply should be sent. The initial value is a null string.

report Feedback information associated with a message. The initial value is MQRO_NONE. The following additional values are possible:

- MQRO_EXCEPTION
- MQRO_EXCEPTION_WITH_DATA
- MQRO_EXCEPTION_WITH_FULL_DATA *
- MQRO_EXPIRATION
- MQRO_EXPIRATION_WITH_DATA
- MQRO_EXPIRATION_WITH_FULL_DATA *
- MQRO_COA

- MQRO_COA_WITH_DATA
- MQRO_COA_WITH_FULL_DATA *
- MQRO_COD
- MQRO_COD_WITH_DATA
- MQRO_COD_WITH_FULL_DATA *
- MQRO_PAN
- MQRO_NAN
- MQRO_NEW_MSG_ID
- MQRO_NEW_CORREL_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID
- MQRO_DEAD_LETTER_Q
- MQRO_DISCARD_MSG

where * indicates values that are not supported on WebSphere MQ for z/OS.

sequence number
Sequence information identifying a message within a group. The initial value is one.

total message length
The number of bytes that were available during the most recent attempt to read a message. This number will be greater than the ImqCache **message length** if the last message was truncated, or if the last message was not read because truncation would have occurred. This attribute is read-only. The initial value is zero.

This attribute can be useful in any situation involving truncated messages.

user id
A user identity associated with a message. The initial value is a null string.

Constructors

ImqMessage( );
The default constructor.

ImqMessage(const ImqMessage & *msg* **);**
The copy constructor. See the **operator =** method for details.

Object methods (public)

void operator = (const ImqMessage & *msg* **);**
Copies the MQMD and message data from *msg*. If a buffer has been supplied by the user for this object, the amount of data copied is restricted to the available buffer size. Otherwise, the system ensures that a buffer of adequate size is made available for the copied data.

ImqString applicationIdData( ) const ;
Returns a copy of the **application id data**.

void setApplicationIdData(const char * *data* **= 0);**
Sets the **application id data**.

ImqString applicationOriginData( ) const ;
Returns a copy of the **application origin data**.

void setApplicationOriginData(const char * *data* **= 0);**
Sets the **application origin data**.

MQLONG backoutCount( ) const ;
Returns the **backout count**.

MQLONG characterSet( ) const ;
Returns the **character set**.

void setCharacterSet(const MQLONG *ccsid* **= MQCCSI_Q_MGR);**
Sets the **character set**.

MQLONG encoding( ) const ;
Returns the **encoding**.

void setEncoding(const MQLONG *encoding* **= MQENC_NATIVE);**
Sets the **encoding**.

MQLONG expiry( ) const ;
Returns the **expiry**.

void setExpiry(const MQLONG *expiry* **);**
Sets the **expiry**.

ImqString format( ) const ;
Returns a copy of the **format**, including trailing blanks.

ImqBoolean formatIs(const char * *format-to-test* **) const ;**
Returns TRUE if the **format** is the same as *format-to-test*.

void setFormat(const char * *name* **= 0);**
Sets the **format**, padded to eight characters with trailing blanks.

MQLONG messageFlags( ) const ;
Returns the **message flags**.

void setMessageFlags(const MQLONG *flags* **);**
Sets the **message flags**.

MQLONG messageType( ) const ;
Returns the **message type**.

void setMessageType(const MQLONG *type* **);**
Sets the **message type**.

MQLONG offset( ) const ;
Returns the **offset**.

void setOffset(const MQLONG *offset* **);**
Sets the **offset**.

MQLONG originalLength( ) const ;
Returns the **original length**.

void setOriginalLength(const MQLONG *length* **);**
Sets the **original length**.

MQLONG persistence( ) const ;
Returns the **persistence**.

void setPersistence(const MQLONG *persistence* **);**
Sets the **persistence**.

MQLONG priority( ) const ;
Returns the **priority**.

void setPriority(const MQLONG *priority* **);**
Sets the **priority**.

ImqString putApplicationName( ) const ;
Returns a copy of the **put application name**.

void setPutApplicationName(const char * *name* = 0);
Sets the **put application name**.

MQLONG putApplicationType( ) const ;
Returns the **put application type**.

void setPutApplicationType(const MQLONG *type* = MQAT_NO_CONTEXT);
Sets the **put application type**.

ImqString putDate( ) const ;
Returns a copy of the **put date**.

void setPutDate(const char * *date* = 0);
Sets the **put date**.

ImqString putTime( ) const ;
Returns a copy of the **put time**.

void setPutTime(const char * *time* = 0);
Sets the **put time**.

ImqBoolean readItem(ImqItem & *item*);
Reads into the *item* object from the message buffer, using the ImqItem **pasteIn** method. It returns TRUE if successful.

ImqString replyToQueueManagerName( ) const ;
Returns a copy of the **reply-to queue manager name**.

void setReplyToQueueManagerName(const char * *name* = 0);
Sets the **reply-to queue manager name**.

ImqString replyToQueueName( ) const ;
Returns a copy of the **reply-to queue name**.

void setReplyToQueueName(const char * *name* = 0);
Sets the **reply-to queue name**.

MQLONG report( ) const ;
Returns the **report**.

void setReport(const MQLONG *report*);
Sets the **report**.

MQLONG sequenceNumber( ) const ;
Returns the **sequence number**.

void setSequenceNumber(const MQLONG *number*);
Sets the **sequence number**.

size_t totalMessageLength( ) const ;
Returns the **total message length**.

ImqString userId( ) const ;
Returns a copy of the **user id**.

void setUserId(const char * *id* = 0);
Sets the **user id**.

ImqBoolean writeItem(ImqItem & *item*);
Writes from the *item* object into the message buffer, using the ImqItem **copyOut** method. Writing can take the form of insertion, replacement, or an append: this depends on the class of the *item* object. This method returns TRUE if successful.

Object methods (protected)

static void setVersionSupported(const MQLONG);
Sets the **MQMD version**. Defaults to **MQMD_VERSION_2**.

Object data (protected)

MQMD1 *omqmd*
(WebSphere MQ for z/OS only.) The MQMD data structure.

MQMD2 *omqmd*
(Platforms other than z/OS.) The MQMD data structure.

ImqMessageTracker C++ class

This class encapsulates those attributes of an ImqMessage or ImqQueue object that can be associated with either object.

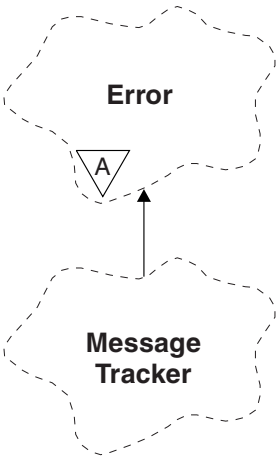


Figure 68. ImqMessageTracker class

This class relates to the MQI calls listed in "ImqMessageTracker cross reference" on page 2646.

Object attributes

accounting token
A binary value (MQBYTE32) of length MQ_ACCOUNTING_TOKEN_LENGTH. The initial value is MQACT_NONE.

correlation id
A binary value (MQBYTE24) of length MQ_CORREL_ID_LENGTH that you assign to correlate messages. The initial value is MQCI_NONE. The additional value, MQCI_NEW_SESSION, is possible.

feedback
Feedback information to be sent with a message. The initial value is MQFB_NONE. The following additional values are possible:

- MQFB_SYSTEM_FIRST
- MQFB_SYSTEM_LAST

- MQFB_APPL_FIRST
- MQFB_APPL_LAST
- MQFB_COA
- MQFB_COD
- MQFB_EXPIRATION
- MQFB_PAN
- MQFB_NAN
- MQFB_QUIT
- MQFB_DATA_LENGTH_ZERO
- MQFB_DATA_LENGTH_NEGATIVE
- MQFB_DATA_LENGTH_TOO_BIG
- MQFB_BUFFER_OVERFLOW
- MQFB_LENGTH_OFF_BY_ONE
- MQFB_IIH_ERROR
- MQFB_NOT_AUTHORIZED_FOR_IMS
- MQFB_IMS_ERROR
- MQFB_IMS_FIRST
- MQFB_IMS_LAST
- MQFB_CICS_APPL_ABENDED
- MQFB_CICS_APPL_NOT_STARTED
- MQFB_CICS_BRIDGE_FAILURE
- MQFB_CICS_CCSID_ERROR
- MQFB_CICS_CIH_ERROR
- MQFB_CICS_COMMAREA_ERROR
- MQFB_CICS_CORREL_ID_ERROR
- MQFB_CICS_DLQ_ERROR
- MQFB_CICS_ENCODING_ERROR
- MQFB_CICS_INTERNAL_ERROR
- MQFB_CICS_NOT_AUTHORIZED
- MQFB_CICS_UOW_BACKED_OUT
- MQFB_CICS_UOW_ERROR

You can also use an application-specific string of your choice. For more information about this, see the "Feedback (MQLONG)" on page 1717 field of the message descriptor (MQMD).

group id
A binary value (MQBYTE24) of length MQ_GROUP_ID_LENGTH unique within a queue. The initial value is MQGI_NONE.

message id
A binary value (MQBYTE24) of length MQ_MSG_ID_LENGTH unique within a queue. The initial value is MQMI_NONE.

Constructors

ImqMessageTracker( );
The default constructor.

ImqMessageTracker(const ImqMessageTracker & *tracker* **);**
The copy constructor. See the **operator =** method for details.

Object methods (public)

void operator = (const ImqMessageTracker & *tracker* **);**
Copies instance data from *tracker*, replacing the existing instance data.

ImqBinary accountingToken( ) const ;
Returns a copy of the **accounting token**.

ImqBoolean setAccountingToken(const ImqBinary & *token* **);**
Sets the **accounting token**. The **data length** of *token* must be either zero or MQ_ACCOUNTING_TOKEN_LENGTH. This method returns TRUE if successful.

void setAccountingToken(const MQBYTE32 *token* **= 0);**
Sets the **accounting token**. *token* can be zero, which is the same as specifying MQACT_NONE. If *token* is nonzero, it must address MQ_ACCOUNTING_TOKEN_LENGTH bytes of binary data. When using predefined values such as MQACT_NONE, you might need to make a cast to ensure a signature match; for example, (MQBYTE *)MQACT_NONE.

ImqBinary correlationId( ) const ;
Returns a copy of the **correlation id**.

ImqBoolean setCorrelationId(const ImqBinary & *token* **);**
Sets the **correlation id**. The **data length** of *token* must be either zero or MQ_CORREL_ID_LENGTH. This method returns TRUE if successful.

void setCorrelationId(const MQBYTE24 *id* **= 0);**
Sets the **correlation id**. *id* can be zero, which is the same as specifying MQCI_NONE. If *id* is nonzero, it must address MQ_CORREL_ID_LENGTH bytes of binary data. When using predefined values such as MQCI_NONE, you might need to make a cast to ensure a signature match; for example, (MQBYTE *)MQCI_NONE.

MQLONG feedback( ) const ;
Returns the **feedback**.

void setFeedback(const MQLONG *feedback* **);**
Sets the **feedback**.

ImqBinary groupId( ) const ;
Returns a copy of the **group id**.

ImqBoolean setGroupId(const ImqBinary & *token* **);**
Sets the **group id**. The **data length** of *token* must be either zero or MQ_GROUP_ID_LENGTH. This method returns TRUE if successful.

void setGroupId(const MQBYTE24 *id* **= 0);**
Sets the **group id**. *id* can be zero, which is the same as specifying MQGI_NONE. If *id* is nonzero, it must address MQ_GROUP_ID_LENGTH bytes of binary data. When using predefined values such as MQGI_NONE, you might need to make a cast to ensure a signature match, for example (MQBYTE *)MQGI_NONE.

ImqBinary messageId( ) const ;
Returns a copy of the **message id**.

ImqBoolean setMessageId(const ImqBinary & *token* **);**
Sets the **message id**. The **data length** of *token* must be either zero or MQ_MSG_ID_LENGTH. This method returns TRUE if successful.

void setMessageId(const MQBYTE24 *id* **= 0);**
Sets the **message id**. *id* can be zero, which is the same as specifying MQMI_NONE. If *id* is nonzero, it must address MQ_MSG_ID_LENGTH bytes of binary data. When using predefined values such as MQMI_NONE, you might need to make a cast to ensure a signature match, for example (MQBYTE *)MQMI_NONE.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR

ImqNamelist C++ class

This class encapsulates a namelist.

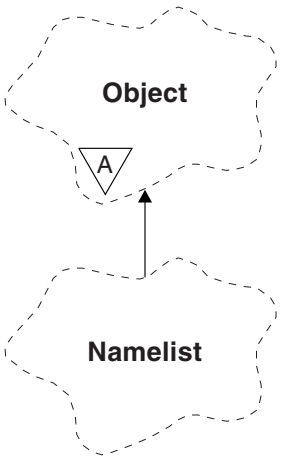


Figure 69. ImqNamelist class

This class relates to the MQI calls listed in "ImqNamelist cross reference" on page 2647.

- "Object attributes"
- "Constructors"
- "Object methods (public)"
- "Reason codes" on page 2699

Object attributes

name count
The number of object names in **namelist names**. This attribute is read-only.

namelist names
Object names, the number of which is indicated by the **name count**. This attribute is read-only.

Constructors

ImqNamelist( );
The default constructor.

ImqNamelist(const ImqNamelist & *list* **);**
The copy constructor. The ImqObject **open status** is false.

ImqNamelist(const char * *name*);
Sets the ImqObject name to **name**.

Object methods (public)

void operator = (const ImqNamelist & *list* **);**
Copies instance data from *list*, replacing the existing instance data. The ImqObject **open status** is false.

ImqBoolean nameCount(MQLONG & *count* **);**
Provides a copy of the **name count**. It returns TRUE if successful.

MQLONG nameCount ( );
Returns the **name count** without any indication of possible errors.

ImqBoolean namelistName (const MQLONG *index***, ImqString &** *name* **);**
Provides a copy of one the **namelist names** by zero based index. It returns TRUE if successful.

ImqString namelistName (const MQLONG *index* **);**
Returns one of the **namelist names** by zero-based index without any indication of possible errors.

Reason codes

- MQRC_INDEX_ERROR
- MQRC_INDEX_NOT_PRESENT

ImqObject C++ class

This class is abstract. When an object of this class is destroyed, it is automatically closed, and its ImqQueueManager connection severed.

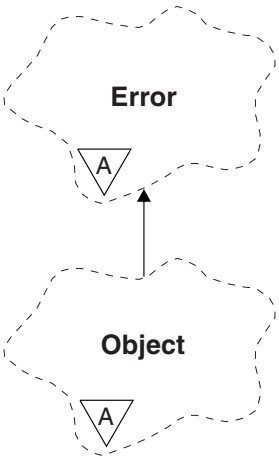


Figure 70. ImqObject class

This class relates to the MQI calls listed in "ImqObject cross reference" on page 2647.

Class attributes

behavior
Controls the behavior of implicit opening.

IMQ_IMPL_OPEN (8L)
Implicit opening is allowed. This is the default.

Object attributes

alteration date
The alteration date. This attribute is read-only.

alteration time
The alteration time. This attribute is read-only.

alternate user id
The alternate user id, up to MQ_USER_ID_LENGTH characters. The initial value is a null string.

alternate security id
The alternate security id. A binary value (MQBYTE40) of length MQ_SECURITY_ID_LENGTH. The initial value is MQSID_NONE.

close options
Options that apply when an object is closed. The initial value is MQCO_NONE. This attribute is ignored during implicit reopen operations, where a value of MQCO_NONE is always used.

connection reference
A reference to an ImqQueueManager object that provides the required connection to a (local) queue manager. For an ImqQueueManager object, it is the object itself. The initial value is zero.

Note: Do not confuse this with the **queue manager name** that identifies a queue manager (possibly remote) for a named queue.

description
The descriptive name (up to 64 characters) of the queue manager, queue, namelist, or process. This attribute is read-only.

name The name (up to 48 characters) of the queue manager, queue, namelist, or process. The initial value is a null string. The name of a model queue changes after an **open** to the name of the resulting dynamic queue.

Note: An ImqQueueManager can have a null name, representing the default queue manager. The name changes to the actual queue manager after a successful **open**. An ImqDistributionList is dynamic and must have a null name.

next managed object
This is the next object of this class, in no particular order, having the same **connection reference** as this object. The initial value is zero.

open options
Options that apply when an object is opened. The initial value is MQOO_INQUIRE. There are two ways to set appropriate values:

1. Do not set the **open options** and do not use the **open** method. WebSphere MQ automatically adjusts the **open options** and automatically opens, reopens, and closes objects as required. This can result in unnecessary reopen operations, because WebSphere MQ uses the **openFor** method, and this adds **open options** incrementally only.
2. Set the **open options** before using any methods that result in an MQI call (see "C++ and MQI cross-reference" on page 2641). This ensures that unnecessary reopen operations do not occur. Set open options explicitly if any of the potential reopen problems are likely to occur (see Reopen).

   If you use the **open** method, you ***must*** ensure that the **open options** are appropriate first. However, using the **open** method is not mandatory; WebSphere MQ still exhibits the same behavior as in case 1, but in this circumstance, the behavior is efficient.

Zero is not a valid value; set the appropriate value before attempting to open the object. This can be done using either **setOpenOptions**(*lOpenOptions*) followed by **open**( ), or **openFor**(*lRequiredOpenOption*).

Note:

1. MQOO_OUTPUT is substituted for MQOO_INQUIRE during the **open** method for a distribution list, as MQOO_OUTPUT is the only valid **open option** at this time. However, it is good practice always to set MQOO_OUTPUT explicitly in application programs that use the **open** method.
2. Specify MQOO_RESOLVE_NAMES if you want to use the **resolved queue manager name** and **resolved queue name** attributes of the class.

open status
Whether the object is open (TRUE) or closed (FALSE). The initial value is FALSE. This attribute is read-only.

previous managed object
The previous object of this class, in no particular order, having the same **connection reference** as this object. The initial value is zero.

queue manager identifier
The queue manager identifier. This attribute is read-only.

Constructors

ImqObject( );
The default constructor.

ImqObject(const ImqObject & *object* **);**
The copy constructor. The **open status** will be FALSE.

Class methods (public)

static MQLONG behavior( );
Returns the **behavior**.

void setBehavior(const MQLONG *behavior* **= 0);**
Sets the **behavior**.

Object methods (public)

void operator = (const ImqObject & *object* **);**
Performs a close if necessary, and copies the instance data from *object*. The **open status** will be FALSE.

ImqBoolean alterationDate(ImqString & *date* **);**
Provides a copy of the **alteration date**. It returns TRUE if successful.

ImqString alterationDate( );
Returns the **alteration date** without any indication of possible errors.

ImqBoolean alterationTime(ImqString & *time* **);**
Provides a copy of the **alteration time**. It returns TRUE if successful.

ImqString alterationTime( );
Returns the **alteration time** without any indication of possible errors.

ImqString alternateUserId( ) const ;
Returns a copy of the **alternate user id**.

ImqBoolean setAlternateUserId(const char * *id* **);**
Sets the **alternate user id**. The **alternate user id** can be set only while the **open status** is FALSE. This method returns TRUE if successful.

ImqBinary alternateSecurityId( ) const ;
Returns a copy of the **alternate security id.**

ImqBoolean setAlternateSecurityId(const ImqBinary & *token*);
Sets the **alternate security id**. The **alternate security id** can be set only while the **open status** is FALSE. The data length of *token* must be either zero or MQ_SECURITY_ID_LENGTH. It returns TRUE if successful.

ImqBoolean setAlternateSecurityId(const MQBYTE* *token* = 0);
Sets the **alternate security id**. *token* can be zero, which is the same as specifying MQSID_NONE. If *token* is nonzero, it must address MQ_SECURITY_ID_LENGTH bytes of binary data. When using predefined values such as MQSID_NONE, you might need to make a cast to ensure signature match; for example, (MQBYTE *)MQSID_NONE.

The **alternate security id** can be set only while the **open status** is TRUE. It returns TRUE if successful.

ImqBoolean setAlternateSecurityId(const unsigned char * *id* = 0);
Sets the **alternate security id**.

ImqBoolean close( );
Sets the **open status** to FALSE. It returns TRUE if successful.

MQLONG closeOptions( ) const ;
Returns the **close options**.

void setCloseOptions(const MQLONG *options*);
Sets the **close options**.

ImqQueueManager * connectionReference( ) const ;
Returns the **connection reference**.

void setConnectionReference(ImqQueueManager & *manager*);
Sets the **connection reference**.

void setConnectionReference(ImqQueueManager * *manager* = 0);
Sets the **connection reference**.

virtual ImqBoolean description(ImqString & *description*) = 0 ;
Provides a copy of the **description**. It returns TRUE if successful.

ImqString description( );
Returns a copy of the **description** without any indication of possible errors.

virtual ImqBoolean name(ImqString & *name*);
Provides a copy of the **name**. It returns TRUE if successful.

ImqString name( );
Returns a copy of the **name** without any indication of possible errors.

ImqBoolean setName(const char * *name* = 0);
Sets the **name**. The **name** can only be set while the **open status** is FALSE, and, for an ImqQueueManager, while the **connection status** is FALSE. It returns TRUE if successful.

ImqObject * nextManagedObject( ) const ;
Returns the **next managed object**.

ImqBoolean open( );
Changes the **open status** to TRUE by opening the object as necessary, using among other attributes the **open options** and the **name**. This method uses the **connection reference** information and the ImqQueueManager **connect** method if necessary to ensure that the ImqQueueManager **connection status** is TRUE. It returns the **open status**.

ImqBoolean openFor(const MQLONG *required-options* = 0);
Attempts to ensure that the object is open with **open options**, or with **open options** that guarantee the behavior implied by the *required-options* parameter value.

If *required-options* is zero, input is required, and any input option suffices. So, if the **open options** already contain one of:

- MQOO_INPUT_AS_Q_DEF
- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE

the **open options** are already satisfactory and are not changed; if the **open options** do not already contain any of these options, MQOO_INPUT_AS_Q_DEF is set in the **open options**.

If *required-options* is nonzero, the required options are added to the **open options**; if *required-options* is any of these options, the others are reset.

If any of the **open options** are changed and the object is already open, the object is closed temporarily and reopened in order to adjust the **open options**.

It returns TRUE if successful. Success indicates that the object is open with appropriate options.

MQLONG openOptions( ) const ;
Returns the **open options**.

ImqBoolean setOpenOptions(const MQLONG *options* **);**
Sets the **open options**. The **open options** can be set only while the **open status** is FALSE. It returns TRUE if successful.

ImqBoolean openStatus( ) const ;
Returns the **open status**.

ImqObject * previousManagedObject( ) const ;
Returns the **previous managed object**.

ImqBoolean queueManagerIdentifier(ImqString & *id* **);**
Provides a copy of the **queue manager identifier**. It returns TRUE if successful.

ImqString queueManagerIdentifier( );
Returns the **queue manager identifier** without any indication of possible errors.

Object methods (protected)

virtual ImqBoolean closeTemporarily( );
Closes an object safely before reopening. It returns TRUE if successful. This method assumes that the **open status** is TRUE.

MQHCONN connectionHandle( ) const ;
Returns the MQHCONN associated with the **connection reference**. This value is zero if there is no **connection reference** or if the Manager is not connected.

ImqBoolean inquire(const MQLONG *int-attr*, **MQLONG &** *value* **);**
Returns an integer value, the index of which is an MQIA_* value. In case of error, the value is set to MQIAV_UNDEFINED.

ImqBoolean inquire(const MQLONG *char-attr*, **char * &** *buffer*, **const size_t** *length* **);**
Returns a character string, the index of which is an MQCA_* value.

Note: Both of these methods return only a single attribute value. If a *snapshot* is required of more than one value, where the values are consistent with each other for an instant, WebSphere MQ C++ does not provide this facility and you must use the MQINQ call with appropriate parameters.

virtual void openInformationDisperse( );
Disperses information from the variable section of the MQOD data structure immediately after an MQOPEN call.

virtual ImqBoolean openInformationPrepare( );
Prepares information for the variable section of the MQOD data structure immediately before an MQOPEN call, and returns TRUE if successful.

ImqBoolean set(const MQLONG *int-attr*, **const MQLONG** *value* **);**
Sets a WebSphere MQ integer attribute.

ImqBoolean set(const MQLONG *char-attr*, **const char *** *buffer*, **const size_t** *required-length* **);**
Sets a WebSphere MQ character attribute.

void setNextManagedObject(const ImqObject * *object* **= 0);**
Sets the **next managed object**.

Attention: Use this function only if you are sure it will not break the managed object list.

void setPreviousManagedObject(const ImqObject * *object* **= 0);**
Sets the **previous managed object**.

Attention: Use this function only if you are sure it will not break the managed object list.

Object data (protected)

MQHOBJ *ohobj*
The WebSphere MQ object handle (valid only when **open status** is TRUE).

MQOD *omqod*
The embedded MQOD data structure. The amount of storage allocated for this data structure is that required for an MQOD Version 2. Inspect the version number *(omqod.Version)* and access the other fields as follows:

MQOD_VERSION_1
All other fields in *omqod* can be accessed.

MQOD_VERSION_2
All other fields in *omqod* can be accessed.

MQOD_VERSION_3
omqod.pmqod is a pointer to a dynamically allocated, larger, MQOD. No other fields in *omqod* can be accessed. All fields addressed by *omqod.pmqod* can be accessed.

Note: *omqod.pmqod.Version* can be less than *omqod.Version*, indicating that the WebSphere MQ MQI client has more functionality than the WebSphere MQ server.

Reason codes

- MQRC_ATTRIBUTE_LOCKED
- MQRC_INCONSISTENT_OBJECT_STATE
- MQRC_NO_CONNECTION_REFERENCE
- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_REOPEN_SAVED_CONTEXT_ERR
- (reason codes from MQCLOSE)
- (reason codes from MQCONN)
- (reason codes from MQINQ)
- (reason codes from MQOPEN)
- (reason codes from MQSET)

ImqProcess C++ class

This class encapsulates an application process (a WebSphere MQ object of type MQOT_PROCESS) that can be triggered by a trigger monitor.

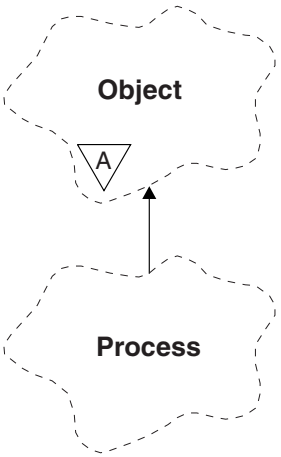


Figure 71. ImqProcess class

- "Object attributes"
- "Constructors"
- "Object methods (public)"

Object attributes

application id
The identity of the application process. This attribute is read-only.

application type
The type of the application process. This attribute is read-only.

environment data
The environment information for the process. This attribute is read-only.

user data
User data for the process. This attribute is read-only.

Constructors

ImqProcess( );
The default constructor.

ImqProcess(const ImqProcess & *process* **);**
The copy constructor. The ImqObject **open status** is FALSE.

ImqProcess(const char * *name* **);**
Sets the ImqObject **name**.

Object methods (public)

void operator = (const ImqProcess & *process* **);**
Performs a close if necessary, and then copies instance data from *process*. The ImqObject **open status** will be FALSE.

ImqBoolean applicationId(ImqString & *id* **);**
Provides a copy of the **application id**. It returns TRUE if successful.

ImqString applicationId( );
Returns the **application id** without any indication of possible errors.

ImqBoolean applicationType(MQLONG & *type* **);**
Provides a copy of the **application type**. It returns TRUE if successful.

MQLONG applicationType( );
Returns the **application type** without any indication of possible errors.

ImqBoolean environmentData(ImqString & *data* **);**
Provides a copy of the **environment data**. It returns TRUE if successful.

ImqString environmentData( );
Returns the **environment data** without any indication of possible errors.

ImqBoolean userData(ImqString & *data* **);**
Provides a copy of the **user data**. It returns TRUE if successful.

ImqString userData( );
Returns the **user data** without any indication of possible errors.

ImqPutMessageOptions C++ class

This class encapsulates the MQPMO data structure.

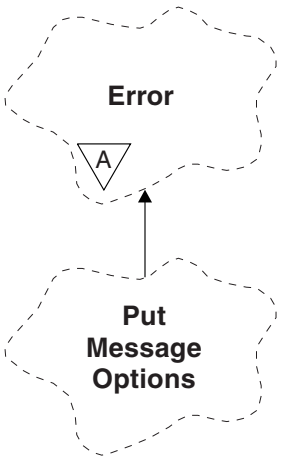


Figure 72. ImqPutMessageOptions class

Object attributes

context reference
An ImqQueue that provides a context for messages. Initially there is no reference.

options
The put message options. The initial value is MQPMO_NONE. The following additional values are possible:

- MQPMO_SYNCPOINT
- MQPMO_NO_SYNCPOINT

- MQPMO_NEW_MSG_ID
- MQPMO_NEW_CORREL_ID
- MQPMO_LOGICAL_ORDER
- MQPMO_NO_CONTEXT
- MQPMO_DEFAULT_CONTEXT
- MQPMO_PASS_IDENTITY_CONTEXT
- MQPMO_PASS_ALL_CONTEXT
- MQPMO_SET_IDENTITY_CONTEXT
- MQPMO_SET_ALL_CONTEXT
- MQPMO_ALTERNATE_USER_AUTHORITY
- MQPMO_FAIL_IF_QUIESCING

record fields
The flags that control the inclusion of put message records when a message is put. The initial value is MQPMRF_NONE. The following additional values are possible:

- MQPMRF_MSG_ID
- MQPMRF_CORREL_ID
- MQPMRF_GROUP_ID
- MQPMRF_FEEDBACK
- MQPMRF_ACCOUNTING_TOKEN

ImqMessageTracker attributes are taken from the object for any field that is specified. ImqMessageTracker attributes are taken from the ImqMessage object for any field that is *not* specified.

resolved queue manager name
Name of a destination queue manager determined during a put. The initial value is null. This attribute is read-only.

resolved queue name
Name of a destination queue determined during a put. The initial value is null. This attribute is read-only.

syncpoint participation
TRUE when messages are put under syncpoint control.

Constructors

ImqPutMessageOptions( );
The default constructor.

ImqPutMessageOptions(const ImqPutMessageOptions & *pmo* **);**
The copy constructor.

Object methods (public)

void operator = (const ImqPutMessageOptions & *pmo* **);**
Copies instance data from *pmo*, replacing the existing instance data.

ImqQueue * contextReference( ) const ;
Returns the **context reference**.

void setContextReference(const ImqQueue & *queue* **);**
Sets the **context reference**.

void setContextReference(const ImqQueue * *queue* **= 0);**
Sets the **context reference**.

MQLONG options( ) const ;
Returns the **options**.

void setOptions(const MQLONG *options* **);**
Sets the **options**, including the **syncpoint participation** value.

MQLONG recordFields( ) const ;
Returns the **record fields**.

void setRecordFields(const MQLONG *fields* **);**
Sets the **record fields**.

ImqString resolvedQueueManagerName( ) const ;
Returns a copy of the **resolved queue manager name**.

ImqString resolvedQueueName( ) const ;
Returns a copy of the **resolved queue name**.

ImqBoolean syncPointParticipation( ) const ;
Returns the **syncpoint participation** value, which is TRUE if the **options** include MQPMO_SYNCPOINT.

void setSyncPointParticipation(const ImqBoolean *sync* **);**
Sets the **syncpoint participation** value. If *sync* is TRUE, the **options** are altered to include MQPMO_SYNCPOINT, and to exclude MQPMO_NO_SYNCPOINT. If *sync* is FALSE, the **options** are altered to include MQPMO_NO_SYNCPOINT, and to exclude MQPMO_SYNCPOINT.

Object data (protected)

MQPMO *omqpmo*
The MQPMO data structure.

Reason codes

- MQRC_STORAGE_NOT_AVAILABLE

ImqQueue C++ class

This class encapsulates a message queue (a WebSphere MQ object of type MQOT_Q).

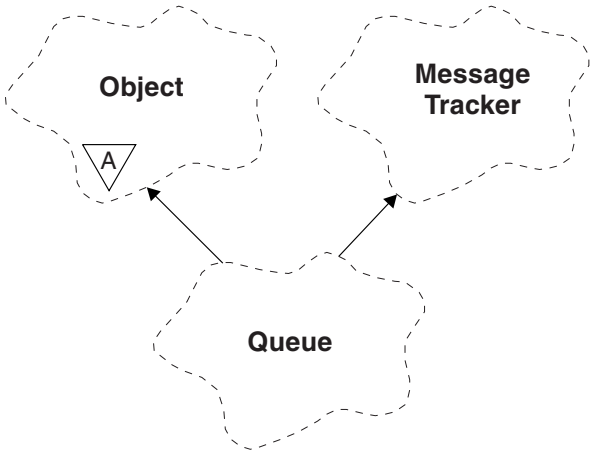


Figure 73. ImqQueue class

This class relates to the MQI calls listed in Table 259 on page 2648.

- "Object attributes" on page 2709
- "Constructors" on page 2712

- "Object methods (public)" on page 2712
- "Object methods (protected)" on page 2719
- "Reason codes" on page 2719

Object attributes

backout requeue name
Excessive backout requeue name. This attribute is read-only.

backout threshold
Backout threshold. This attribute is read-only.

base queue name
Name of the queue that the alias resolves to. This attribute is read-only.

cluster name
Cluster name. This attribute is read-only.

cluster namelist name
Cluster namelist name. This attribute is read-only.

cluster workload rank
Cluster workload rank. This attribute is read-only.

cluster workload priority
Cluster workload priority. This attribute is read-only.

cluster workload use queue
Cluster workload use queue value. This attribute is read-only.

creation date
Queue creation data. This attribute is read-only.

creation time
Queue creation time. This attribute is read-only.

current depth
Number of messages on the queue. This attribute is read-only.

default bind
Default bind. This attribute is read-only.

default input open option
Default open-for-input option. This attribute is read-only.

default persistence
Default message persistence. This attribute is read-only.

default priority
Default message priority. This attribute is read-only.

definition type
Queue definition type. This attribute is read-only.

depth high event
Control attribute for queue depth high events. This attribute is read-only.

depth high limit
High limit for the queue depth. This attribute is read-only.

depth low event
Control attribute for queue depth low events. This attribute is read-only.

depth low limit
Low limit for the queue depth. This attribute is read-only.

depth maximum event
Control attribute for queue depth maximum events. This attribute is read-only.

distribution list reference
Optional reference to an ImqDistributionList that can be used to distribute messages to more than one queue, including this one. The initial value is null.

Note: When an ImqQueue object is opened, any open ImqDistributionList object that it references is automatically closed.

distribution lists
The capability of a transmission queue to support distribution lists. This attribute is read-only.

dynamic queue name
Dynamic queue name. The initial value is AMQ.* for all Windows, UNIX, and Linux platforms.

harden get backout
Whether to harden the backout count. This attribute is read-only.

index type
Index type. This attribute is read-only.

inhibit get
Whether get operations are allowed. The initial value is dependent on the queue definition. This attribute is valid for an alias or local queue only.

inhibit put
Whether put operations are allowed. The initial value is dependent on the queue definition.

initiation queue name
Name of the initiation queue. This attribute is read-only.

maximum depth
Maximum number of messages allowed on the queue. This attribute is read-only.

maximum message length
Maximum length for any message on this queue, which can be less than the maximum for any queue managed by the associated queue manager. This attribute is read-only.

message delivery sequence
Whether message priority is relevant. This attribute is read-only.

next distributed queue
Next object of this class, in no particular order, having the same **distribution list reference** as this object. The initial value is zero.

If an object in a chain is deleted, the previous object and next object are updated so that their distributed queue links no longer point to the deleted object.

non-persistent message class
Level of reliability for non-persistent messages put to this queue. This attribute is read-only.

open input count
Number of ImqQueue objects that are open for input. This attribute is read-only.

open output count
Number of ImqQueue objects that are open for output. This attribute is read-only.

previous distributed queue
Previous object of this class, in no particular order, having the same **distribution list reference** as this object. The initial value is zero.

If an object in a chain is deleted, the previous object and next object are updated so that their distributed queue links no longer point to the deleted object.

process name
Name of the process definition. This attribute is read-only.

queue accounting
Level of accounting information for queues. This attribute is read-only.

queue manager name
Name of the queue manager (possibly remote) where the queue resides. Do not confuse the queue manager named here with the ImqObject **connection reference**, which references the (local) queue manager providing a connection. The initial value is null.

queue monitoring
Level of monitoring data collection for the queue. This attribute is read-only.

queue statistics
Level of statistics data for the queue. This attribute is read-only.

queue type
Queue type. This attribute is read-only.

remote queue manager name
Name of the remote queue manager. This attribute is read-only.

remote queue name
Name of the remote queue as known on the remote queue manager. This attribute is read-only.

resolved queue manager name
Resolved queue manager name. This attribute is read-only.

resolved queue name
Resolved queue name. This attribute is read-only.

retention interval
Queue retention interval. This attribute is read-only.

scope Scope of the queue definition. This attribute is read-only.

service interval
Service interval. This attribute is read-only.

service interval event
Control attribute for service interval events. This attribute is read-only.

shareability
Whether the queue can be shared. This attribute is read-only.

storage class
Storage class. This attribute is read-only.

transmission queue name
Name of the transmission queue. This attribute is read-only.

trigger control
Trigger control. The initial value depends on the queue definition. This attribute is valid for a local queue only.

trigger data
Trigger data. The initial value depends on the queue definition. This attribute is valid for a local queue only.

trigger depth
Trigger depth. The initial value depends on the queue definition. This attribute is valid for a local queue only.

trigger message priority
Threshold message priority for triggers. The initial value depends on the queue definition. This attribute is valid for a local queue only.

trigger type
Trigger type. The initial value depends on the queue definition. This attribute is valid for a local queue only.

usage Usage. This attribute is read-only.

Constructors

ImqQueue( );
The default constructor.

ImqQueue(const ImqQueue & *queue* **);**
The copy constructor. The ImqObject **open status** will be FALSE.

ImqQueue(const char * *name* **);**
Sets the ImqObject **name**.

Object methods (public)

void operator = (const ImqQueue & *queue* **);**
Performs a close if necessary, and then copies instance data from *queue*. The ImqObject **open status** will be FALSE.

ImqBoolean backoutRequeueName(ImqString & *name* **);**
Provides a copy of the **backout requeue name**. It returns TRUE if successful.

ImqString backoutRequeueName( );
Returns the **backout requeue name** without any indication of possible errors.

ImqBoolean backoutThreshold(MQLONG & *threshold* **);**
Provides a copy of the **backout threshold**. It returns TRUE if successful.

MQLONG backoutThreshold( );
Returns the **backout threshold** value without any indication of possible errors.

ImqBoolean baseQueueName(ImqString & *name* **);**
Provides a copy of the **base queue name**. It returns TRUE if successful.

ImqString baseQueueName( );
Returns the **base queue name** without any indication of possible errors.

ImqBoolean clusterName(ImqString & *name* **);**
Provides a copy of the **cluster name**. It returns TRUE if successful.

ImqString clusterName( );
Returns the **cluster name** without any indication of possible errors.

ImqBoolean clusterNamelistName(ImqString & *name* **);**
Provides a copy of the **cluster namelist name**. It returns TRUE if successful.

ImqString clusterNamelistName( );
Returns the **cluster namelist name** without any indication of errors.

ImqBoolean clusterWorkLoadPriority (MQLONG & priority);
Provides a copy of the cluster workload priority value. It returns TRUE if successful.

MQLONG clusterWorkLoadPriority ( );
Returns the cluster workload priority value without any indication of possible errors.

ImqBoolean clusterWorkLoadRank (MQLONG & rank);
Provides a copy of the cluster workload rank value. It returns TRUE if successful.

MQLONG clusterWorkLoadRank ( );
Returns the cluster workload rank value without any indication of possible errors.

ImqBoolean clusterWorkLoadUseQ (MQLONG & useq);
Provides a copy of the cluster workload use queue value. It returns TRUE if successful.

MQLONG clusterWorkLoadUseQ ( );
Returns the cluster workload use queue value without any indication of possible errors.

ImqBoolean creationDate(ImqString & *date* **);**
Provides a copy of the **creation date**. It returns TRUE if successful.

ImqString creationDate( );
Returns the **creation date** without any indication of possible errors.

ImqBoolean creationTime(ImqString & *time* **);**
Provides a copy of the **creation time**. It returns TRUE if successful.

ImqString creationTime( );
Returns the **creation time** without any indication of possible errors.

ImqBoolean currentDepth(MQLONG & *depth* **);**
Provides a copy of the **current depth**. It returns TRUE if successful.

MQLONG currentDepth( );
Returns the **current depth** without any indication of possible errors.

ImqBoolean defaultInputOpenOption(MQLONG & *option* **);**
Provides a copy of the **default input open option**. It returns TRUE if successful.

MQLONG defaultInputOpenOption( );
Returns the **default input open option** without any indication of possible errors.

ImqBoolean defaultPersistence(MQLONG & *persistence* **);**
Provides a copy of the **default persistence**. It returns TRUE if successful.

MQLONG defaultPersistence( );
Returns the **default persistence** without any indication of possible errors.

ImqBoolean defaultPriority(MQLONG & *priority* **);**
Provides a copy of the **default priority**. It returns TRUE if successful.

MQLONG defaultPriority( );
Returns the **default priority** without any indication of possible errors.

ImqBoolean defaultBind(MQLONG & *bind* **);**
Provides a copy of the **default bind**. It returns TRUE if successful.

MQLONG defaultBind( );
Returns the **default bind** without any indication of possible errors.

ImqBoolean definitionType(MQLONG & *type* **);**
Provides a copy of the **definition type**. It returns TRUE if successful.

MQLONG definitionType( );
Returns the **definition type** without any indication of possible errors.

ImqBoolean depthHighEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the **depth high event**. It returns TRUE if successful.

MQLONG depthHighEvent( );
Returns the enablement state of the **depth high event** without any indication of possible errors.

ImqBoolean depthHighLimit(MQLONG & *limit* **);**
Provides a copy of the **depth high limit**. It returns TRUE if successful.

MQLONG depthHighLimit( );
Returns the **depth high limit** value without any indication of possible errors.

ImqBoolean depthLowEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the **depth low event**. It returns TRUE if successful.

MQLONG depthLowEvent( );
Returns the enablement state of the **depth low event** without any indication of possible errors.

ImqBoolean depthLowLimit(MQLONG & *limit* **);**
Provides a copy of the **depth low limit**. It returns TRUE if successful.

MQLONG depthLowLimit( );
Returns the **depth low limit** value without any indication of possible errors.

ImqBoolean depthMaximumEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the **depth maximum event**. It returns TRUE if successful.

MQLONG depthMaximumEvent( );
Returns the enablement state of the **depth maximum event** without any indication of possible errors.

ImqDistributionList * distributionListReference( ) const ;
Returns the **distribution list reference**.

void setDistributionListReference(ImqDistributionList & *list* **);**
Sets the **distribution list reference**.

void setDistributionListReference(ImqDistributionList * *list* **= 0);**
Sets the **distribution list reference**.

ImqBoolean distributionLists(MQLONG & *support* **);**
Provides a copy of the **distribution lists** value. It returns TRUE if successful.

MQLONG distributionLists( );
Returns the **distribution lists** value without any indication of possible errors.

ImqBoolean setDistributionLists(const MQLONG *support* **);**
Sets the **distribution lists** value. It returns TRUE if successful.

ImqString dynamicQueueName( ) const ;
Returns a copy of the **dynamic queue name**.

ImqBoolean setDynamicQueueName(const char * *name* **);**
Sets the **dynamic queue name**. The **dynamic queue name** can be set only while the ImqObject **open status** is FALSE. It returns TRUE if successful.

ImqBoolean get(ImqMessage & *msg*, **ImqGetMessageOptions &** *options* **);**
Retrieves a message from the queue, using the specified *options*. Invokes the ImqObject **openFor** method if necessary to ensure that the ImqObject **open options** include either one of the MQOO_INPUT_* values, or the MQOO_BROWSE value, depending on the *options*. If the *msg* object has an ImqCache **automatic buffer**, the buffer grows to accommodate any message retrieved. The **clearMessage** method is invoked against the *msg* object before retrieval.

This method returns TRUE if successful.

Note: The result of the method invocation is FALSE if the ImqObject **reason code** is MQRC_TRUNCATED_MSG_FAILED, even though this **reason code** is classified as a warning. If a truncated message is accepted, the ImqCache **message length** reflects the truncated length. In either event, the ImqMessage **total message length** indicates the number of bytes that were available.

ImqBoolean get(ImqMessage & *msg* **);**
As for the previous method, except that default get message options are used.

ImqBoolean get(ImqMessage & *msg*, **ImqGetMessageOptions &** *options*, **const size_t** *buffer-size* **);**
As for the previous two methods, except that an overriding *buffer-size* is indicated. If the *msg* object employs an ImqCache **automatic buffer**, the **resizeBuffer** method is invoked on the *msg* object prior to message retrieval, and the buffer does not grow further to accommodate any larger message.

ImqBoolean get(ImqMessage & *msg*, **const size_t** *buffer-size* **);**
As for the previous method, except that default get message options are used.

ImqBoolean hardenGetBackout(MQLONG & *harden* **);**
Provides a copy of the **harden get backout** value. It returns TRUE if successful.

MQLONG hardenGetBackout( );
Returns the **harden get backout** value without any indication of possible errors.

ImqBoolean indexType(MQLONG & *type* **);**
Provides a copy of the **index type**. It returns TRUE if successful.

MQLONG indexType( );
Returns the **index type** without any indication of possible errors.

ImqBoolean inhibitGet(MQLONG & *inhibit* **);**
Provides a copy of the **inhibit get** value. It returns TRUE if successful.

MQLONG inhibitGet( );
Returns the **inhibit get** value without any indication of possible errors.

ImqBoolean setInhibitGet(const MQLONG *inhibit* **);**
Sets the **inhibit get** value. It returns TRUE if successful.

ImqBoolean inhibitPut(MQLONG & *inhibit* **);**
Provides a copy of the **inhibit put** value. It returns TRUE if successful.

MQLONG inhibitPut( );
Returns the **inhibit put** value without any indication of possible errors.

ImqBoolean setInhibitPut(const MQLONG *inhibit* **);**
Sets the **inhibit put** value. It returns TRUE if successful.

ImqBoolean initiationQueueName(ImqString & *name* **);**
Provides a copy of the **initiation queue name**. It returns TRUE if successful.

ImqString initiationQueueName( );
Returns the **initiation queue name** without any indication of possible errors.

ImqBoolean maximumDepth(MQLONG & *depth* **);**
Provides a copy of the **maximum depth**. It returns TRUE if successful.

MQLONG maximumDepth( );
Returns the **maximum depth** without any indication of possible errors.

ImqBoolean maximumMessageLength(MQLONG & *length* **);**
Provides a copy of the **maximum message length**. It returns TRUE if successful.

MQLONG maximumMessageLength( );
Returns the **maximum message length** without any indication of possible errors.

ImqBoolean messageDeliverySequence(MQLONG & *sequence* **);**
Provides a copy of the **message delivery sequence**. It returns TRUE if successful.

MQLONG messageDeliverySequence( );
Returns the **message delivery sequence** value without any indication of possible errors.

ImqQueue * nextDistributedQueue( ) const ;
Returns the **next distributed queue**.

ImqBoolean nonPersistentMessageClass (MQLONG & monq);
Provides a copy of the non persistent message class value. It returns TRUE if successful.

MQLONG nonPersistentMessageClass ( );
Returns the non persistent message class value without any indication of possible errors.

ImqBoolean openInputCount(MQLONG & *count* **);**
Provides a copy of the **open input count**. It returns TRUE if successful.

MQLONG openInputCount( );
Returns the **open input count** without any indication of possible errors.

ImqBoolean openOutputCount(MQLONG & *count* **);**
Provides a copy of the **open output count**. It returns TRUE if successful.

MQLONG openOutputCount( );
Returns the **open output count** without any indication of possible errors.

ImqQueue * previousDistributedQueue( ) const ;
Returns the **previous distributed queue**.

ImqBoolean processName(ImqString & *name* **);**
Provides a copy of the **process name**. It returns TRUE if successful.

ImqString processName( );
Returns the **process name** without any indication of possible errors.

ImqBoolean put(ImqMessage & *msg* **);**
Places a message onto the queue, using default put message options. Uses the ImqObject **openFor** method if necessary to ensure that the ImqObject **open options** include MQOO_OUTPUT.

This method returns TRUE if successful.

ImqBoolean put(ImqMessage & *msg*, **ImqPutMessageOptions &** *pmo* **);**
Places a message onto the queue, using the specified *pmo*. Uses the ImqObject **openFor** method as necessary to ensure that the ImqObject **open options** include MQOO_OUTPUT, and (if the *pmo* **options** include any of MQPMO_PASS_IDENTITY_CONTEXT, MQPMO_PASS_ALL_CONTEXT, MQPMO_SET_IDENTITY_CONTEXT, or MQPMO_SET_ALL_CONTEXT) corresponding MQOO_*_CONTEXT values.

This method returns TRUE if successful.

Note: If the *pmo* includes a **context reference**, the referenced object is opened, if necessary, to provide a context.

ImqBoolean queueAccounting (MQLONG & acctq);
Provides a copy of the queue accounting value. It returns TRUE if successful.

MQLONG queueAccounting ( );
Returns the queue accounting value without any indication of possible errors.

ImqString queueManagerName( ) const ;
Returns the **queue manager name**.

ImqBoolean setQueueManagerName(const char * *name* **);**
Sets the **queue manager name**. The **queue manager name** can be set only while the ImqObject **open status** is FALSE. This method returns TRUE if successful.

ImqBoolean queueMonitoring (MQLONG & monq);
Provides a copy of the queue monitoring value. It returns TRUE if successful.

MQLONG queueMonitoring ( );
Returns the queue monitoring value without any indication of possible errors.

ImqBoolean queueStatistics (MQLONG & statq);
Provides a copy of the queue statistics value. It returns TRUE if successful.

MQLONG queueStatistics ( );
Returns the queue statistics value without any indication of possible errors.

ImqBoolean queueType(MQLONG & *type* **);**
Provides a copy of the **queue type** value. It returns TRUE if successful.

MQLONG queueType( );
Returns the **queue type** without any indication of possible errors.

ImqBoolean remoteQueueManagerName(ImqString & *name* **);**
Provides a copy of the **remote queue manager name**. It returns TRUE if successful.

ImqString remoteQueueManagerName( );
Returns the **remote queue manager name** without any indication of possible errors.

ImqBoolean remoteQueueName(ImqString & *name* **);**
Provides a copy of the **remote queue name**. It returns TRUE if successful.

ImqString remoteQueueName( );
Returns the **remote queue name** without any indication of possible errors.

ImqBoolean resolvedQueueManagerName(ImqString & *name* **);**
Provides a copy of the **resolved queue manager name**. It returns TRUE if successful.

Note: This method fails unless MQOO_RESOLVE_NAMES is among the ImqObject **open options**.

ImqString resolvedQueueManagerName( ) ;
Returns the **resolved queue manager name**, without any indication of possible errors.

ImqBoolean resolvedQueueName(ImqString & *name* **);**
Provides a copy of the **resolved queue name**. It returns TRUE if successful.

Note: This method fails unless MQOO_RESOLVE_NAMES is among the ImqObject **open options**.

ImqString resolvedQueueName( ) ;
Returns the **resolved queue name**, without any indication of possible errors.

ImqBoolean retentionInterval(MQLONG & *interval* **);**
Provides a copy of the **retention interval**. It returns TRUE if successful.

MQLONG retentionInterval( );
Returns the **retention interval** without any indication of possible errors.

ImqBoolean scope(MQLONG & *scope* **);**
Provides a copy of the **scope**. It returns TRUE if successful.

MQLONG scope( );
Returns the **scope** without any indication of possible errors.

ImqBoolean serviceInterval(MQLONG & *interval* **);**
Provides a copy of the **service interval**. It returns TRUE if successful.

MQLONG serviceInterval( );
Returns the **service interval** without any indication of possible errors.

ImqBoolean serviceIntervalEvent(MQLONG & *event*);
Provides a copy of the enablement state of the **service interval event**. It returns TRUE if successful.

MQLONG serviceIntervalEvent( );
Returns the enablement state of the **service interval event** without any indication of possible errors.

ImqBoolean shareability(MQLONG & *shareability*);
Provides a copy of the **shareability** value. It returns TRUE if successful.

MQLONG shareability( );
Returns the **shareability** value without any indication of possible errors.

ImqBoolean storageClass(ImqString & *class*);
Provides a copy of the **storage class**. It returns TRUE if successful.

ImqString storageClass( );
Returns the **storage class** without any indication of possible errors.

ImqBoolean transmissionQueueName(ImqString & *name*);
Provides a copy of the **transmission queue name**. It returns TRUE if successful.

ImqString transmissionQueueName( );
Returns the **transmission queue name** without any indication of possible errors.

ImqBoolean triggerControl(MQLONG & *control*);
Provides a copy of the **trigger control** value. It returns TRUE if successful.

MQLONG triggerControl( );
Returns the **trigger control** value without any indication of possible errors.

ImqBoolean setTriggerControl(const MQLONG *control*);
Sets the **trigger control** value. It returns TRUE if successful.

ImqBoolean triggerData(ImqString & *data*);
Provides a copy of the **trigger data**. It returns TRUE if successful.

ImqString triggerData( );
Returns a copy of the **trigger data** without any indication of possible errors.

ImqBoolean setTriggerData(const char * *data*);
Sets the **trigger data**. It returns TRUE if successful.

ImqBoolean triggerDepth(MQLONG & *depth*);
Provides a copy of the **trigger depth**. It returns TRUE if successful.

MQLONG triggerDepth( );
Returns the **trigger depth** without any indication of possible errors.

ImqBoolean setTriggerDepth(const MQLONG *depth*);
Sets the **trigger depth**. It returns TRUE if successful.

ImqBoolean triggerMessagePriority(MQLONG & *priority*);
Provides a copy of the **trigger message priority**. It returns TRUE if successful.

MQLONG triggerMessagePriority( );
Returns the **trigger message priority** without any indication of possible errors.

ImqBoolean setTriggerMessagePriority(const MQLONG *priority*);
Sets the **trigger message priority**. It returns TRUE if successful.

ImqBoolean triggerType(MQLONG & *type*);
Provides a copy of the **trigger type**. It returns TRUE if successful.

MQLONG triggerType( );
Returns the **trigger type** without any indication of possible errors.

ImqBoolean setTriggerType(const MQLONG *type* **);**
Sets the **trigger type**. It returns TRUE if successful.

ImqBoolean usage(MQLONG & *usage* **);**
Provides a copy of the **usage** value. It returns TRUE if successful.

MQLONG usage( );
Returns the **usage** value without any indication of possible errors.

Object methods (protected)

void setNextDistributedQueue(ImqQueue * *queue* **= 0);**
Sets the **next distributed queue**.

Attention: Use this function only if you are sure it will not break the distributed queue list.

void setPreviousDistributedQueue(ImqQueue * *queue* **= 0);**
Sets the **previous distributed queue**.

Attention: Use this function only if you are sure it will not break the distributed queue list.

Reason codes

- MQRC_ATTRIBUTE_LOCKED
- MQRC_CONTEXT_OBJECT_NOT_VALID
- MQRC_CONTEXT_OPEN_ERROR
- MQRC_CURSOR_NOT_VALID
- MQRC_NO_BUFFER
- MQRC_REOPEN_EXCL_INPUT_ERROR
- MQRC_REOPEN_INQUIRE_ERROR
- MQRC_REOPEN_TEMPORARY_Q_ERROR
- (reason codes from MQGET)
- (reason codes from MQPUT)

ImqQueueManager C++ class

This class encapsulates a queue manager (a WebSphere MQ object of type MQOT_Q_MGR).

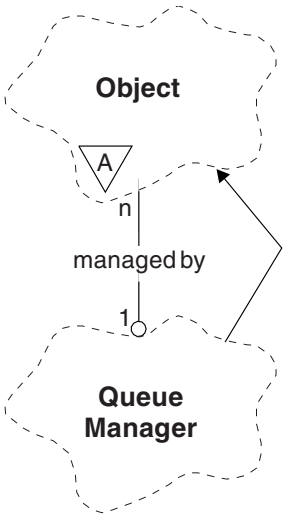


Figure 74. ImqQueueManager class

This class relates to the MQI calls listed in "ImqQueueManager cross-reference" on page 2650. Not all the listed methods are applicable to all platforms; see ALTER QMGR for more details.

Class attributes

behavior

Controls the behavior of implicit connection and disconnection.

IMQ_EXPL_DISC_BACKOUT (0L)
An explicit call to the **disconnect** method implies backout. This attribute is mutually exclusive with IMQ_EXPL_DISC_COMMIT.

IMQ_EXPL_DISC_COMMIT (1L)
An explicit call to the **disconnect** method implies commit (the default). This attribute is mutually exclusive with IMQ_EXPL_DISC_BACKOUT.

IMQ_IMPL_CONN (2L)
Implicit connection is allowed (the default).

IMQ_IMPL_DISC_BACKOUT (0L)
An implicit call to the **disconnect** method, which can occur during object destruction, implies backout. This attribute is mutually exclusive with the IMQ_IMPL_DISC_COMMIT.

IMQ_IMPL_DISC_COMMIT (4L)
An implicit call to the **disconnect** method, which can occur during object destruction, implies commit (the default). This attribute is mutually exclusive with IMQ_IMPL_DISC_BACKOUT.

At WebSphere MQ V7.0 and above, C++ applications that make use of an implicit connection, need to specify IMQ_IMPL_CONN along with any other options provided in the `setBehavior()` method on an object of class `ImqQueueManager`. If your application does not use the `setBehavior()` method to explicitly set the behavior options, for example,

```
ImqQueueManager_object.setBehavior(IMQ_IMPL_DISC_COMMIT)
```

this change does not affect you since MQ_IMPL_CONN is enabled by default.

If your application explicitly sets the behavior options, for example,

```
ImqQueueManager_object.setBehavior(IMQ_IMPL_DISC_COMMIT)
```

you need to include IMQ_IMPL_CONN in the `setBehavior()` method as follows, to allow your application to complete an implicit connection:

```
ImqQueueManager_object.setBehavior(IMQ_IMPL_CONN | IMQ_IMPL_DISC_COMMIT)
```

Object attributes

accounting connections override
Allows applications to override the setting of the MQI accounting and queue accounting values.This attribute is read-only.

accounting interval
How long before intermediate accounting records are written (in seconds). This attribute is read-only.

activity recording
Controls the generation of activity reports. This attribute is read-only.

adopt new mca check
The elements checked to determine if an MCA should be adopted when a new inbound channel is detected that has the same name as an MCA that is already active. This attribute is read-only.

adopt new mca type
Whether an orphaned instance of an MCA of a particular channel type should be restarted automatically when a new inbound channel request matching the adopt new mca check parameters is detected. This attribute is read-only.

authentication type
Indicates the type of authentication which is being performed.

authority event
Controls authority events. This attribute is read-only.

begin options
Options that apply to the **begin** method. The initial value is MQBO_NONE.

bridge event
Whether IMS Bridge events are generated. This attribute is read-only.

channel auto definition
Channel auto definition value. This attribute is read-only.

channel auto definition event
Channel auto definition event value. This attribute is read-only.

channel auto definition exit
Channel auto definition exit name. This attribute is read-only.

channel event
Whether channel events are generated. This attribute is read-only.

channel initiator adapters
The number of adapter subtasks to use for processing WebSphere MQ calls. This attribute is read-only.

channel initiator control
Whether the Channel Initiator should be started automatically when the Queue Manager is started. This attribute is read-only.

channel initiator dispatchers
The number of dispatchers to use for the channel initiator. This attribute is read-only.

channel initiator trace autostart
Whether channel initiator trace should start automatically or not. This attribute is read-only.

channel initiator trace table size
The size of the channel initiator's trace data space (in MB). This attribute is read-only.

channel monitoring
Controls the collection of online monitoring data for channels. This attribute is read-only.

channel reference
A reference to a channel definition for use during client connection. While connected, this attribute can be set to null, but cannot be changed to any other value. The initial value is null.

channel statistics
Controls the collection of statistics data for channels. This attribute is read-only.

character set
Coded character set identifier (CCSID). This attribute is read-only.

cluster sender monitoring
Controls the collection of online monitoring data for automatically-defined cluster sender channels. This attribute is read-only.

cluster sender statistics
Controls the collection of statistics data for automatically defined cluster sender channels. This attribute is read-only.

cluster workload data
Cluster workload exit data. This attribute is read-only.

cluster workload exit
Cluster workload exit name. This attribute is read-only.

cluster workload length
Cluster workload length. This attribute is read-only.

cluster workload mru
Cluster workload most recently used channels value. This attribute is read-only.

cluster workload use queue
Cluster workload use queue value. This attribute is read-only.

command event
Whether command events are generated. This attribute is read-only.

command input queue name
System command input queue name. This attribute is read-only.

command level
Command level supported by the queue manager. This attribute is read-only.

command server control
Whether the Command Server should be started automatically when the Queue Manager is started. This attribute is read-only.

connect options
Options that apply to the **connect** method. The initial value is MQCNO_NONE. The following additional values may be possible, depending on platform:
- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_HANDLE_SHARE_NONE
- MQCNO_HANDLE_SHARE_BLOCK
- MQCNO_HANDLE_SHARE_NO_BLOCK
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_QSG

connection id
A unique identifier that allows MQ to reliably identify an application.

connection status
TRUE when connected to the queue manager. This attribute is read-only.

connection tag
A tag to be associated with a connection. This attribute can only be set when not connected. The initial value is null.

cryptographic hardware
Configuration details for cryptographic hardware. For MQ MQI client connections.

dead-letter queue name
Name of the dead-letter queue. This attribute is read-only.

default transmission queue name
Default transmission queue name. This attribute is read-only.

distribution lists
Capability of the queue manager to support distribution lists.

dns group
The name of the group that the TCP listener that handles inbound transmissions for the queue-sharing group should join when using Workload Manager Dynamic Domain Name Services support. This attribute is read-only.

dns wlm
Whether the TCP listener that handles inbound transmissions for the queue-sharing group should register with Workload Manager for Dynamic Domain Name Services. This attribute is read-only.

first authentication record
The first of one or more objects of class ImqAuthenticationRecord, in no particular order, in which the ImqAuthenticationRecord connection reference addresses this object. For MQ MQI client connections.

first managed object
The first of one or more objects of class ImqObject, in no particular order, in which the ImqObject **connection reference** addresses this object. The initial value is zero.

inhibit event
Controls inhibit events. This attribute is read-only.

ip address version
Which IP protocol (IPv4 or IPv6) to use for a channel connection. This attribute is read-only.

key repository
Location of the key database file in which keys and certificates are stored. For WebSphere MQ MQI client connections.

key reset count
The number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated. This attribute applies only to client connections using MQCONNX. See also ssl key reset count.

listener timer
The time interval (in seconds) between attempts by WebSphere MQ to restart the listener if there has been an APPC or TCP/IP failure. This attribute is read-only.

local event
Controls local events. This attribute is read-only.

logger event
Controls whether recovery log events are generated. This attribute is read-only.

lu group name
The generic LU name that the LU 6.2 listener that handles inbound transmissions for the queue-sharing group should use. This attribute is read-only.

lu name
The name of the LU to use for outbound LU 6.2 transmissions. This attribute is read-only.

lu62 arm suffix
The suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator. This attribute is read-only.

lu62 channels
The maximum number of channels that can be current or clients that can be connected, that use the LU 6.2 transmission protocol. This attribute is read-only.

maximum active channels
The maximum number of channels that can be active at any time. This attribute is read-only.

maximum channels
The maximum number of channels that can be current (including server-connection channels with connected clients). This attribute is read-only.

maximum handles
Maximum number of handles. This attribute is read-only.

maximum message length
Maximum possible length for any message on any queue managed by this queue manager. This attribute is read-only.

maximum priority
Maximum message priority. This attribute is read-only.

maximum uncommitted messages
Maximum number of uncommitted messages within a unit or work. This attribute is read-only.

mqi accounting
Controls the collection of accounting information for MQI data. This attribute is read-only.

mqi statistics
Controls the collection of statistics monitoring information for the queue manager. This attribute is read-only.

outbound port maximum
The higher end of the range of port numbers to be used when binding outgoing channels. This attribute is read-only.

outbound port minimum
The lower end of the range of port numbers to be used when binding outgoing channels. This attribute is read-only.

password
password associated with user ID

performance event
Controls performance events. This attribute is read-only.

platform
Platform on which the queue manager resides. This attribute is read-only.

queue accounting
Controls the collection of accounting information for queues. This attribute is read-only.

queue monitoring
Controls the collection of online monitoring data for queues. This attribute is read-only.

queue statistics
Controls the collection of statistics data for queues. This attribute is read-only.

receive timeout
Approximately how long a TCP/IP message channel will wait to receive data, including heartbeats, from its partner, before returning to the inactive state. This attribute is read-only.

receive timeout minimum
The minimum time that a TCP/IP channel will wait to receive data, including heartbeats, from its partner, before returning to the inactive state. This attribute is read-only.

receive timeout type
A qualifier applied to **receive timeout**. This attribute is read-only.

remote event
Controls remote events. This attribute is read-only.

repository name
Repository name. This attribute is read-only.

repository namelist
Repository namelist name. This attribute is read-only.

shared queue manager name
Whether MQOPENs of a shared queue where the ObjectQMgrName is another queue manager in the queue-sharing group should resolve to an open of the shared queue on the local queue manager. This attribute is read-only.

ssl event
Whether SSL events are generated. This attribute is read-only.

ssl FIPS required
Whether only FIPS-certified algorithms should be used if the cryptography is executed in WebSphere MQ software. This attribute is read-only.

ssl key reset count
The number of unencrypted bytes sent and received within an SSL conversation before the secret key is renegotiated. This attribute is read-only.

start-stop event
Controls start-stop events. This attribute is read-only.

statistics interval
How often statistics monitoring data is written to the monitoring queue. This attribute is read-only.

syncpoint availability
Availability of syncpoint participation. This attribute is read-only.

Note: Queue manager-coordinated global units of work are not supported on the IBM i platform.

tcp channels
The maximum number of channels that can be current or clients that can be connected, that use the TCP/IP transmission protocol. This attribute is read-only.

tcp keepalive
Whether the TCP KEEPALIVE facility is to be used to check that the other end of the connection is still available. This attribute is read-only.

tcp name
The name of either the sole or default TCP/IP system to be used, depending on the value of **tcp stack type**. This attribute is read-only.

tcp stack type
Whether the channel initiator is permitted to only use the TCP/IP address space specified in **tcp name** or can bind to any selected TCP/IP address. This attribute is read-only.

trace route recording
Controls the recording of route tracing information. This attribute is read-only.

trigger interval
Trigger interval. This attribute is read-only.

user id
On UNIX and Linux platforms, the application's real user ID. On Windowsplatforms, the application's user ID.

Constructors

ImqQueueManager( );
The default constructor.

ImqQueueManager(const ImqQueueManager & *manager* **);**
The copy constructor. The **connection status** will be FALSE.

ImqQueueManager(const char * *name* **);**
Sets the ImqObject **name** to *name*.

Destructors

When an ImqQueueManager object is destroyed, it is automatically disconnected.

Class methods (public)

static MQLONG behavior( );
Returns the **behavior**.

void setBehavior(const MQLONG *behavior* **= 0);**
Sets the **behavior**.

Object methods (public)

void operator = (const ImqQueueManager & *mgr* **);**
Disconnects if necessary, and copies instance data from *mgr*. The **connection status** is be FALSE.

ImqBoolean accountingConnOverride (MQLONG & statint);
Provides a copy of the accounting connections override value. It returns TRUE if successful.

MQLONG accountingConnOverride ( );
Returns the accounting connections override value without any indication of possible errors.

ImqBoolean accountingInterval (MQLONG & statint);
Provides a copy of the accounting interval value. It returns TRUE if successful.

MQLONG accountingInterval ( );
Returns the accounting interval value without any indication of possible errors.

ImqBoolean activityRecording (MQLONG & rec);
Provides a copy of the activity recording value. It returns TRUE if successful.

MQLONG activityRecording ( );
Returns the activity recording value without any indication of possible errors.

ImqBoolean adoptNewMCACheck (MQLONG & check);
Provides a copy of the adopt new MCA check value. It returns TRUE if successful.

MQLONG adoptNewMCACheck ( );
Returns the adopt new MCA check value without any indication of possible errors.

ImqBoolean adoptNewMCAType (MQLONG & type);
Provides a copy of the adopt new MCA type. It returns TRUE if successful.

MQLONG adoptNewMCAType ( );
Returns the adopt new MCA type without any indication of possible errors.

QLONG authenticationType ( ) const;
Returns the authentication type.

void setAuthenticationType (const MQLONG type = MQCSP_AUTH_NONE);
Sets the authentication type.

ImqBoolean authorityEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the **authority event**. It returns TRUE if successful.

MQLONG authorityEvent( );
Returns the enablement state of the **authority event** without any indication of possible errors.

ImqBoolean backout( );
Backs out uncommitted changes. It returns TRUE if successful.

ImqBoolean begin( );
Begins a unit of work. The **begin options** affect the behavior of this method. It returns TRUE if successful, but it also returns TRUE even if the underlying MQBEGIN call returns MQRC_NO_EXTERNAL_PARTICIPANTS or MQRC_PARTICIPANT_NOT_AVAILABLE (which are both associated with MQCC_WARNING).

MQLONG beginOptions( ) const ;
Returns the **begin options**.

void setBeginOptions(const MQLONG *options* **= MQBO_NONE);**
Sets the **begin options**.

ImqBoolean bridgeEvent (MQLONG & event);
Provides a copy of the bridge event value. It returns TRUE if successful.

MQLONG bridgeEvent ( );
Returns the bridge event value without any indication of possible errors.

ImqBoolean channelAutoDefinition(MQLONG & *value* **);**
Provides a copy of the **channel auto definition** value. It returns TRUE if successful.

MQLONG channelAutoDefinition( );
Returns the **channel auto definition** value without any indication of possible errors.

ImqBoolean channelAutoDefinitionEvent(MQLONG & *value* **);**
Provides a copy of the **channel auto definition event** value. It returns TRUE if successful.

MQLONG channelAutoDefinitionEvent( );
Returns the **channel auto definition event** value without any indication of possible errors.

ImqBoolean channelAutoDefinitionExit(ImqString & *name* **);**
Provides a copy of the **channel auto definition exit** name. It returns TRUE if successful.

ImqString channelAutoDefinitionExit( );
Returns the **channel auto definition exit** name without any indication of possible errors.

ImqBoolean channelEvent (MQLONG & event);
Provides a copy of the channel event value. It returns TRUE if successful.

MQLONG channelEvent( );
Returns the channel event value without any indication of possible errors.

MQLONG channelInitiatorAdapters ( );
Returns the channel initiator adapters value without any indication of possible errors.

ImqBoolean channelInitiatorAdapters (MQLONG & adapters);
Provides a copy of the channel initiator adapters value. It returns TRUE if successful.

MQLONG channelInitiatorControl ( );
Returns the channel initiator startup value without any indication of possible errors.

ImqBoolean channelInitiatorControl (MQLONG & init);
Provides a copy of the channel initiator control startup value. It returns TRUE if successful.

MQLONG channelInitiatorDispatchers ( );
Returns the channel initiator dispatchers value without any indication of possible errors.

ImqBoolean channelInitiatorDispatchers (MQLONG & dispatchers);
Provides a copy of the channel initiator dispatchers value. It returns TRUE if successful.

MQLONG channelInitiatorTraceAutoStart ( );
Returns the channel initiator trace auto start value without any indication of possible errors.

ImqBoolean channelInitiatorTraceAutoStart (MQLONG & auto);
Provides a copy of the channel initiator trace auto start value. It returns TRUE if successful.

MQLONG channelInitiatorTraceTableSize ( );
Returns the channel initiator trace table size value without any indication of possible errors.

ImqBoolean channelInitiatorTraceTableSize (MQLONG & size);
Provides a copy of the channel initiator trace table size value. It returns TRUE if successful.

ImqBoolean channelMonitoring (MQLONG & monchl);
Provides a copy of the channel monitoring value. It returns TRUE if successful.

MQLONG channelMonitoring ( );
Returns the channel monitoring value without any indication of possible errors.

ImqBoolean channelReference(ImqChannel * & *pchannel* **);**
Provides a copy of the **channel reference**. If the **channel reference** is invalid, sets *pchannel* to null. This method returns TRUE if successful.

ImqChannel * channelReference( );
Returns the **channel reference** without any indication of possible errors.

ImqBoolean setChannelReference(ImqChannel & *channel* **);**
Sets the **channel reference**. This method returns TRUE if successful.

ImqBoolean setChannelReference(ImqChannel * *channel* **= 0);**
Sets or resets the **channel reference**. This method returns TRUE if successful.

ImqBoolean channelStatistics (MQLONG & statchl);
Provides a copy of the channel statistics value. It returns TRUE if successful.

MQLONG channelStatistics ( );
Returns the channel statistics value without any indication of possible errors.

ImqBoolean characterSet(MQLONG & *ccsid* **);**
Provides a copy of the **character set**. It returns TRUE if successful.

MQLONG characterSet( );
Returns a copy of the **character set**, without any indication of possible errors.

MQLONG clientSslKeyResetCount ( ) const;
Returns the SSL key reset count value used on client connections.

void setClientSslKeyResetCount(const MQLONG count);
Sets the SSL key reset count used on client connections.

ImqBoolean clusterSenderMonitoring (MQLONG & monacls);
Provides a copy of the cluster sender monitoring default value. It returns TRUE if successful.

MQLONG clusterSenderMonitoring ( );
Returns the cluster sender monitoring default value without any indication of possible errors.

ImqBoolean clusterSenderStatistics (MQLONG & statacls);
Provides a copy of the cluster sender statistics value. It returns TRUE if successful.

MQLONG clusterSenderStatistics ( );
Returns the cluster sender statistics value without any indication of possible errors.

ImqBoolean clusterWorkloadData(ImqString & *data* **);**
Provides a copy of the **cluster workload exit data**. It returns TRUE if successful.

ImqString clusterWorkloadData( );
Returns the **cluster workload exit data** without any indication of possible errors.

ImqBoolean clusterWorkloadExit(ImqString & *name* **);**
Provides a copy of the **cluster workload exit name**. It returns TRUE if successful.

ImqString clusterWorkloadExit( );
Returns the **cluster workload exit name** without any indication of possible errors.

ImqBoolean clusterWorkloadLength(MQLONG & *length* **);**
Provides a copy of the **cluster workload length**. It returns TRUE if successful.

MQLONG clusterWorkloadLength( );
Returns the **cluster workload length** without any indication of possible errors.

ImqBoolean clusterWorkLoadMRU (MQLONG & mru);
Provides a copy of the cluster workload most recently used channels value. It returns TRUE if successful.

MQLONG clusterWorkLoadMRU ( );
Returns the cluster workload most recently used channels value without any indication of possible errors.

ImqBoolean clusterWorkLoadUseQ (MQLONG & useq);
Provides a copy of the cluster workload use queue value. It returns TRUE if successful.

MQLONG clusterWorkLoadUseQ ( );
Returns the cluster workload use queue value without any indication of possible errors.

ImqBoolean commandEvent (MQLONG & event);
Provides a copy of the command event value. It returns TRUE if successful.

MQLONG commandEvent ( );
Returns the command event value without any indication of possible errors.

ImqBoolean commandInputQueueName(ImqString & *name* **);**
Provides a copy of the **command input queue name**. It returns TRUE if successful.

ImqString commandInputQueueName( );
Returns the **command input queue name** without any indication of possible errors.

ImqBoolean commandLevel(MQLONG & *level* **);**
Provides a copy of the **command level**. It returns TRUE if successful.

MQLONG commandLevel( );
Returns the **command level** without any indication of possible errors.

MQLONG commandServerControl ( );
Returns the command server startup value without any indication of possible errors.

ImqBoolean commandServerControl (MQLONG & server);
Provides a copy of the command server control startup value. It returns TRUE if successful.

ImqBoolean commit( );
Commits uncommitted changes. It returns TRUE if successful.

ImqBoolean connect( );
Connects to the queue manager with the given ImqObject **name**, the default being the local queue manager. If you want to connect to a specific queue manager, use the ImqObject **setName** method before connection. If there is a **channel reference**, it is used to pass information about the channel definition to MQCONNX in an MQCD. The ChannelType in the MQCD is set to MQCHT_CLNTCONN. **channel reference** information, which is only meaningful for client connections, is ignored for server connections. The **connect options** affect the behavior of this method. This method sets the **connection status** to TRUE if successful. It returns the new connection status.

If there is a first authentication record, the chain of authentication records is used to authenticate digital certificates for secure client channels.

You can connect more than one ImqQueueManager object to the same queue manager. All use the same MQHCONN connection handle and share UOW functionality for the connection associated with the thread. The first ImqQueueManager to connect obtains the MQHCONN handle. The last ImqQueueManager to disconnect performs the MQDISC.

For a multithreaded program, it is recommended that a separate ImqQueueManager object is used for each thread.

ImqBinary connectionId ( ) const ;
Returns the connection ID.

ImqBinary connectionTag ( ) const ;
Returns the **connection tag**.

ImqBoolean setConnectionTag (const MQBYTE128 *tag* **= 0);**
Sets the **connection tag**. If *tag* is zero, clears the **connection tag**. This method returns TRUE if successful.

ImqBoolean setConnectionTag (const ImqBinary & *tag* **);**
Sets the **connection tag**. The **data length** of *tag* must be either zero (to clear the **connection tag**) or MQ_CONN_TAG_LENGTH. This method returns TRUE if successful.

MQLONG connectOptions( ) const ;
Returns the **connect options**.

void setConnectOptions(const MQLONG *options* **= MQCNO_NONE);**
Sets the **connect options**.

ImqBoolean connectionStatus( ) const ;
Returns the **connection status**.

ImqString cryptographicHardware ( );
Returns the **cryptographic hardware**.

ImqBoolean setCryptographicHardware (const char * *hardware* = 0);
Sets the **cryptographic hardware**. This method returns TRUE if successful.

ImqBoolean deadLetterQueueName(ImqString & *name*);
Provides a copy of the **dead-letter queue name**. It returns TRUE if successful.

ImqString deadLetterQueueName( );
Returns a copy of the **dead-letter queue name**, without any indication of possible errors.

ImqBoolean defaultTransmissionQueueName(ImqString & *name*);
Provides a copy of the **default transmission queue name**. It returns TRUE if successful.

ImqString defaultTransmissionQueueName( );
Returns the **default transmission queue name** without any indication of possible errors.

ImqBoolean disconnect( );
Disconnects from the queue manager and sets the **connection status** to FALSE. Closes all ImqProcess and ImqQueue objects associated with this object, and severs their **connection reference** before disconnection. If more than one ImqQueueManager object is connected to the same queue manager, only the last to disconnect performs a physical disconnection; others perform a logical disconnection. Uncommitted changes are committed on physical disconnection only.

This method returns TRUE if successful. If it is called when there is no existing connection, the return code is also true.

ImqBoolean distributionLists(MQLONG & *support*);
Provides a copy of the **distribution lists** value. It returns TRUE if successful.

MQLONG distributionLists( );
Returns the **distribution lists** value without any indication of possible errors.

ImqBoolean dnsGroup (ImqString & group);
Provides a copy of the DNS group name. It returns TRUE if successful.

ImqString dnsGroup ( );
Returns the DNS group name without any indication of possible errors.

ImqBoolean dnsWlm (MQLONG & wlm);
Provides a copy of the DNS WLM value. It returns TRUE if successful.

MQLONG dnsWlm ( );
Returns the DNS WLM value without any indication of possible errors.

ImqAuthenticationRecord * firstAuthenticationRecord ( ) const ;
Returns the **first authentication record**.

void setFirstAuthenticationRecord (const ImqAuthenticationRecord * *air* = 0);
Sets the **first authentication record**.

ImqObject * firstManagedObject( ) const ;
Returns the **first managed object**.

ImqBoolean inhibitEvent(MQLONG & *event*);
Provides a copy of the enablement state of the **inhibit event**. It returns TRUE if successful.

MQLONG inhibitEvent( );
Returns the enablement state of the **inhibit event** without any indication of possible errors.

ImqBoolean ipAddressVersion (MQLONG & version);
Provides a copy of the IP address version value. It returns TRUE if successful.

MQLONG ipAddressVersion ( );
Returns the IP address version value without any indication of possible errors.

ImqBoolean keepAlive (MQLONG & keepalive);
Provides a copy of the keep alive value. It returns TRUE if successful.

MQLONG keepAlive ( );
Returns the keep alive value without any indication of possible errors.

ImqString keyRepository ( );
Returns the **key repository**.

ImqBoolean setKeyRepository (const char * *repository* **= 0);**
Sets the **key repository**. It returns TRUE if successful.

ImqBoolean listenerTimer (MQLONG & timer);
Provides a copy of the listener timer value. It returns TRUE if successful.

MQLONG listenerTimer ( );
Returns the listener timer value without any indication of possible errors.

ImqBoolean localEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the **local event**. It returns TRUE if successful.

MQLONG localEvent( );
Returns the enablement state of the **local event** without any indication of possible errors.

ImqBoolean loggerEvent (MQLONG & count);
Provides a copy of the logger event value. It returns TRUE if successful.

MQLONG loggerEvent ( );
Returns the logger event value without any indication of possible errors.

ImqBoolean luGroupName (ImqString & name);.
Provides a copy of the LU group name. It returns TRUE if successful

ImqString luGroupName ( );
Returns the LU group name without any indication of possible errors.

ImqBoolean lu62ARMSuffix (ImqString & suffix);
Provides a copy of the LU62 ARM suffix. It returns TRUE if successful.

ImqString lu62ARMSuffix ( );
Returns the LU62 ARM suffix without any indication of possible errors

ImqBoolean luName (ImqString & name);
Provides a copy of the LU name. It returns TRUE if successful.

ImqString luName ( );
Returns the LU name without any indication of possible errors.

ImqBoolean maximumActiveChannels (MQLONG & channels);
Provides a copy of the maximum active channels value. It returns TRUE if successful.

MQLONG maximumActiveChannels ( );
Returns the maximum active channels value without any indication of possible errors.

ImqBoolean maximumCurrentChannels (MQLONG & channels);
Provides a copy of the maximum current channels value. It returns TRUE if successful.

MQLONG maximumCurrentChannels ( );
Returns the maximum current channels value without any indication of possible errors.

ImqBoolean maximumHandles(MQLONG & *number* **);**
Provides a copy of the **maximum handles**. It returns TRUE if successful.

MQLONG maximumHandles( );
Returns the **maximum handles** without any indication of possible errors.

ImqBoolean maximumLu62Channels (MQLONG & channels);
Provides a copy of the maximum LU62 channels value. It returns TRUE if successful.

MQLONG maximumLu62Channels ( );.
Returns the maximum LU62 channels value without any indication of possible errors

ImqBoolean maximumMessageLength(MQLONG & *length* **);**
Provides a copy of the **maximum message length**. It returns TRUE if successful.

MQLONG maximumMessageLength( );
Returns the **maximum message length** without any indication of possible errors.

ImqBoolean maximumPriority(MQLONG & *priority* **);**
Provides a copy of the **maximum priority**. It returns TRUE if successful.

MQLONG maximumPriority( );
Returns a copy of the **maximum priority**, without any indication of possible errors.

ImqBoolean maximumTcpChannels (MQLONG & channels);
Provides a copy of the maximum TCP channels value. It returns TRUE if successful.

MQLONG maximumTcpChannels ( );
Returns the maximum TCP channels value without any indication of possible errors.

ImqBoolean maximumUncommittedMessages(MQLONG & *number* **);**
Provides a copy of the **maximum uncommitted messages**. It returns TRUE if successful.

MQLONG maximumUncommittedMessages( );
Returns the **maximum uncommitted messages** without any indication of possible errors.

ImqBoolean mqiAccounting (MQLONG & statint);
Provides a copy of the MQI accounting value. It returns TRUE if successful.

MQLONG mqiAccounting ( );
Returns the MQI accounting value without any indication of possible errors.

ImqBoolean mqiStatistics (MQLONG & statmqi);
Provides a copy of the MQI statistics value. It returns TRUE if successful.

MQLONG mqiStatistics ( );
Returns the MQI statistics value without any indication of possible errors.

ImqBoolean outboundPortMax (MQLONG & max);
Provides a copy of the maximum outbound port value. It returns TRUE if successful.

MQLONG outboundPortMax ( );
Returns the maximum outbound port value without any indication of possible errors.

ImqBoolean outboundPortMin (MQLONG & min);
Provides a copy of the minimum outbound port value. It returns TRUE if successful.

MQLONG outboundPortMin ( );
Returns the minimum outbound port value without any indication of possible errors.

ImqBinary password ( ) const;
Returns the password used on client connections.

ImqBoolean setPassword (const ImqString & password);
Sets the password used on client connections.

ImqBoolean setPassword (const char * = 0 password);
Sets the password used on client connections.

ImqBoolean setPassword (const ImqBinary & password);
Sets the password used on client connections.

ImqBoolean performanceEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the **performance event**. It returns TRUE if successful.

MQLONG performanceEvent( );
Returns the enablement state of the **performance event** without any indication of possible errors.

ImqBoolean platform(MQLONG & *platform* **);**
Provides a copy of the **platform**. It returns TRUE if successful.

MQLONG platform( );
Returns the **platform** without any indication of possible errors.

ImqBoolean queueAccounting (MQLONG & acctq);
Provides a copy of the queue accounting value. It returns TRUE if successful.

MQLONG queueAccounting ( );
Returns the queue accounting value without any indication of possible errors.

ImqBoolean queueMonitoring (MQLONG & monq);
Provides a copy of the queue monitoring value. It returns TRUE if successful.

MQLONG queueMonitoring ( );
Returns the queue monitoring value without any indication of possible errors.

ImqBoolean queueStatistics (MQLONG & statq);
Provides a copy of the queue statistics value. It returns TRUE if successful.

MQLONG queueStatistics ( );
Returns the queue statistics value without any indication of possible errors.

ImqBoolean receiveTimeout (MQLONG & timeout);
Provides a copy of the receive timeout value. It returns TRUE if successful.

MQLONG receiveTimeout ( );
Returns the receive timeout value without any indication of possible errors.

ImqBoolean receiveTimeoutMin (MQLONG & min);
Provides a copy of the minimum receive timeout value. It returns TRUE if successful.

MQLONG receiveTimeoutMin ( );
Returns the minimum receive timeout value without any indication of possible errors.

ImqBoolean receiveTimeoutType (MQLONG & type);
Provides a copy of the receive timeout type. It returns TRUE if successful.

MQLONG receiveTimeoutType ( );
Returns the receive timeout type without any indication of possible errors.

ImqBoolean remoteEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the **remote event**. It returns TRUE if successful.

MQLONG remoteEvent( );
Returns the enablement state of the **remote event** without any indication of possible errors.

ImqBoolean repositoryName(ImqString & *name* **);**
Provides a copy of the **repository name**. It returns TRUE if successful.

ImqString repositoryName( );
Returns the **repository name** without any indication of possible errors.

ImqBoolean repositoryNamelistName(ImqString & *name* **);**
Provides a copy of the **repository namelist name**. It returns TRUE if successful.

ImqString repositoryNamelistName( );
Returns a copy of the **repository namelist name** without any indication of possible errors.

ImqBoolean sharedQueueQueueManagerName (MQLONG & name);
Provides a copy of the shared queue queue manager name value. It returns TRUE if successful.

MQLONG sharedQueueQueueManagerName ( );
Returns the shared queue queue manager name value without any indication of possible errors.

ImqBoolean sslEvent (MQLONG & event);
Provides a copy of the SSL event value. It returns TRUE if successful.

MQLONG sslEvent ( );
Returns the SSL event value without any indication of possible errors.

ImqBoolean sslFips (MQLONG & sslfips);
Provides a copy of the SSL FIPS value. It returns TRUE if successful.

MQLONG sslFips ( );
Returns the SSL FIPS value without any indication of possible errors.

ImqBoolean sslKeyResetCount (MQLONG & count);
Provides a copy of the SSL key reset count value. It returns TRUE if successful.

MQLONG sslKeyResetCount ( );
Returns the SSL key reset count value without any indication of possible errors.

ImqBoolean startStopEvent(MQLONG & *event* **);**
Provides a copy of the enablement state of the **start-stop event**. It returns TRUE if successful.

MQLONG startStopEvent( );
Returns the enablement state of the **start-stop event** without any indication of possible errors.

ImqBoolean statisticsInterval (MQLONG & statint);
Provides a copy of the statistics interval value. It returns TRUE if successful.

MQLONG statisticsInterval ( );
Returns the statistics interval value without any indication of possible errors.

ImqBoolean syncPointAvailability(MQLONG & *sync* **);**
Provides a copy of the **syncpoint availability** value. It returns TRUE if successful.

MQLONG syncPointAvailability( );
Returns a copy of the **syncpoint availability** value, without any indication of possible errors.

ImqBoolean tcpName (ImqString & name);
Provides a copy of the TCP system name. It returns TRUE if successful.

ImqString tcpName ( );
Returns the TCP system name without any indication of possible errors.

ImqBoolean tcpStackType (MQLONG & type);
Provides a copy of the TCP stack type. It returns TRUE if successful.

MQLONG tcpStackType ( );
Returns the TCP stack type without any indication of possible errors.

ImqBoolean traceRouteRecording (MQLONG & routerec);
Provides a copy of the trace route recording value. It returns TRUE if successful.

MQLONG traceRouteRecording ( );
Returns the trace route recording value without any indication of possible errors.

ImqBoolean triggerInterval(MQLONG & *interval* **);**
Provides a copy of the **trigger interval**. It returns TRUE if successful.

MQLONG triggerInterval( );
Returns the **trigger interval** without any indication of possible errors.

ImqBinary userId ( ) const;
Returns the user ID used on client connections.

ImqBoolean setUserId (const ImqString & id);
Sets the user ID used on client connections.

ImqBoolean setUserId (const char * = 0 id);
Sets the user ID used on client connections.

ImqBoolean setUserId (const ImqBinary & id);
Sets the user ID used on client connections.

Object methods (protected)

void setFirstManagedObject(const ImqObject * *object* **= 0);**
Sets the **first managed object**.

Object data (protected)

MQHCONN *ohconn*
The WebSphere MQ connection handle (meaningful only while the **connection status** is TRUE).

Reason codes

- MQRC_ATTRIBUTE_LOCKED
- MQRC_ENVIRONMENT_ERROR
- MQRC_FUNCTION_NOT_SUPPORTED
- MQRC_REFERENCE_ERROR
- (reason codes for MQBACK)
- (reason codes for MQBEGIN)
- (reason codes for MQCMIT)
- (reason codes for MQCONNX)
- (reason codes for MQDISC)
- (reason codes for MQCONN)

ImqReferenceHeader C++ class

This class encapsulates features of the MQRMH data structure.

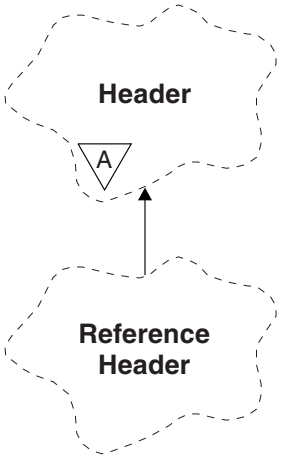


Figure 75. ImqReferenceHeader class

This class relates to the MQI calls listed in "ImqReferenceHeader cross-reference" on page 2653.

Object attributes

destination environment
Environment for the destination. The initial value is a null string.

destination name
Name of the data destination. The initial value is a null string.

instance id
Instance identifier. A binary value (MQBYTE24) of length MQ_OBJECT_INSTANCE_ID_LENGTH. The initial value is MQOII_NONE.

logical length
Logical, or intended, length of message data that follows this header. The initial value is zero.

logical offset
Logical offset for the message data that follows, to be interpreted in the context of the data as a whole, at the ultimate destination. The initial value is zero.

logical offset 2
High-order extension to the **logical offset**. The initial value is zero.

reference type
Reference type. The initial value is a null string.

source environment
Environment for the source. The initial value is a null string.

source name
Name of the data source. The initial value is a null string.

Constructors

ImqReferenceHeader( );
The default constructor.

ImqReferenceHeader(const ImqReferenceHeader & *header* **);**
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg* **);**
Inserts an MQRMH data structure into the message buffer at the beginning, moving existing message data further along, and sets the *msg* **format** to MQFMT_REF_MSG_HEADER.

See the ImqHeader class method description on "ImqHeader C++ class" on page 2682 for further details.

virtual ImqBoolean pasteIn(ImqMessage & *msg* **);**
Reads an MQRMH data structure from the message buffer.

To be successful, the ImqMessage **format** must be MQFMT_REF_MSG_HEADER.

See the ImqHeader class method description on "ImqHeader C++ class" on page 2682 for further details.

Object methods (public)

void operator = (const ImqReferenceHeader & *header*);
Copies instance data from *header*, replacing the existing instance data.

ImqString destinationEnvironment( ) const ;
Returns a copy of the **destination environment**.

void setDestinationEnvironment(const char * *environment* = 0);
Sets the **destination environment**.

ImqString destinationName( ) const ;
Returns a copy of the **destination name**.

void setDestinationName(const char * *name* = 0);
Sets the **destination name**.

ImqBinary instanceId( ) const ;
Returns a copy of the **instance id**.

ImqBoolean setInstanceId(const ImqBinary & *id*);
Sets the **instance id**. The **data length** of *token* must be either 0 or MQ_OBJECT_INSTANCE_ID_LENGTH. This method returns TRUE if successful.

void setInstanceId(const MQBYTE24 *id* = 0);
Sets the **instance id**. *id* can be zero, which is the same as specifying MQOII_NONE. If *id* is nonzero, it must address MQ_OBJECT_INSTANCE_ID_LENGTH bytes of binary data. When using pre-defined values such as MQOII_NONE, you might need to make a cast to ensure a signature match, for example (MQBYTE *)MQOII_NONE.

MQLONG logicalLength( ) const ;
Returns the **logical length**.

void setLogicalLength(const MQLONG *length*);
Sets the **logical length**.

MQLONG logicalOffset( ) const ;
Returns the **logical offset**.

void setLogicalOffset(const MQLONG *offset*);
Sets the **logical offset**.

MQLONG logicalOffset2( ) const ;
Returns the **logical offset 2**.

void setLogicalOffset2(const MQLONG *offset*);
Sets the **logical offset 2**.

ImqString referenceType( ) const ;
Returns a copy of the **reference type**.

void setReferenceType(const char * *name* = 0);
Sets the **reference type**.

ImqString sourceEnvironment( ) const ;
Returns a copy of the **source environment**.

void setSourceEnvironment(const char * *environment* = 0);
Sets the **source environment**.

ImqString sourceName( ) const ;
Returns a copy of the **source name**.

void setSourceName(const char * *name* = 0);
Sets the **source name**.

Object data (protected)

MQRMH *omqrmh*
The MQRMH data structure.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR
- MQRC_STRUC_LENGTH_ERROR
- MQRC_STRUC_ID_ERROR
- MQRC_INSUFFICIENT_DATA
- MQRC_INCONSISTENT_FORMAT
- MQRC_ENCODING_ERROR

ImqString C++ class

This class provides character string storage and manipulation for null-terminated strings.

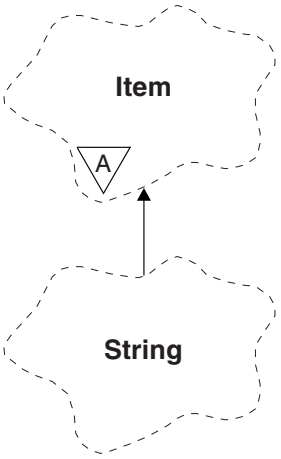


Figure 76. ImqString class

Use an ImqString in place of a **char *** in most situations where a parameter calls for a **char ***.

- "Object attributes"
- "Constructors" on page 2740
- "Class methods (public)" on page 2740
- "Overloaded ImqItem methods" on page 2740
- "Object methods (public)" on page 2740
- "Object methods (protected)" on page 2743
- "Reason codes" on page 2743

Object attributes

characters
Characters in the **storage** that precede a trailing null.

length Number of bytes in the **characters**. If there is no **storage**, the **length** is zero. The initial value is zero.

storage
A volatile array of bytes of arbitrary size. A trailing null must always be present in the **storage** after the **characters**, so that the end of the **characters** can be detected. Methods ensure that this

situation is maintained, but ensure, when setting bytes in the array directly, that a trailing null exists after modification. Initially, there is no **storage** attribute.

Constructors

ImqString( );
The default constructor.

ImqString(const ImqString & *string* **);**
The copy constructor.

ImqString(const char *c* **);**
The **characters** comprise *c*.

ImqString(const char * *text* **);**
The **characters** are copied from *text*.

ImqString(const void * *buffer*, **const size_t** *length* **);**
Copies *length* bytes starting from *buffer* and assigns them to the **characters**. Substitution is made for any null characters copied. The substitution character is a period (.). No special consideration is given to any other non-printable or non-displayable characters copied.

Class methods (public)

static ImqBoolean copy(char * *destination-buffer*, **const size_t** *length*, **const char *** *source-buffer*, **const char** *pad* **= 0);**
Copies up to *length* bytes from *source-buffer* to *destination-buffer*. If the number of characters in *source-buffer* is insufficient, fills the remaining space in *destination-buffer* with *pad* characters. *source-buffer* can be zero. *destination-buffer* can be zero if *length* is also zero. Any error codes are lost. This method returns TRUE if successful.

static ImqBoolean copy (char * *destination-buffer*, **const size_t** *length*, **const char *** *source-buffer*, **ImqError &** *error-object*, **const char** *pad* **= 0);**
Copies up to *length* bytes from *source-buffer* to *destination-buffer*. If the number of characters in *source-buffer* is insufficient, fills the remaining space in *destination-buffer* with *pad* characters. *source-buffer* can be zero. *destination-buffer* can be zero if *length* is also zero. Any error codes are set in *error-object*. This method returns TRUE if successful.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg* **);**
Copies the **characters** to the message buffer, replacing any existing content. Sets the *msg* **format** to MQFMT_STRING.

See the parent class method description for further details.

virtual ImqBoolean pasteIn(ImqMessage & *msg* **);**
Sets the **characters** by transferring the remaining data from the message buffer, replacing the existing **characters**.

To be successful, the **encoding** of the *msg* object must be MQENC_NATIVE. Retrieve messages with MQGMO_CONVERT to MQENC_NATIVE.

To be successful, the ImqMessage **format** must be MQFMT_STRING.

See the parent class method description for further details.

Object methods (public)

char & operator [ ] (const size_t *offset* **) const ;**
References the character at offset *offset* in the **storage**. Ensure that the relevant byte exists and is addressable.

ImqString operator ( ) (const size_t *offset*, **const size_t** *length* **= 1) const ;**
Returns a substring by copying bytes from the **characters** starting at *offset*. If *length* is zero, returns the rest of the **characters**. If the combination of *offset* and *length* does not produce a reference within the **characters**, returns an empty ImqString.

void operator = (const ImqString & *string* **);**
Copies instance data from *string*, replacing the existing instance data.

ImqString operator + (const char *c* **) const ;**
Returns the result of appending *c* to the **characters**.

ImqString operator + (const char * *text* **) const ;**
Returns the result of appending *text* to the **characters**. This can also be inverted. For example:

```
strOne + "string two" ;
"string one" + strTwo ;
```

Note: Although most compilers accept **strOne + "string two";** Microsoft Visual C++ requires **strOne + (char *)"string two" ;**

ImqString operator + (const ImqString & *string1* **) const ;**
Returns the result of appending *string1* to the **characters**.

ImqString operator + (const double *number* **) const ;**
Returns the result of appending *number* to the **characters** after conversion to text.

ImqString operator + (const long *number* **) const ;**
Returns the result of appending *number* to the **characters** after conversion to text.

void operator += (const char *c* **);**
Appends *c* to the **characters**.

void operator += (const char * *text* **);**
Appends *text* to the **characters**.

void operator += (const ImqString & *string* **);**
Appends *string* to the **characters**.

void operator += (const double *number* **);**
Appends *number* to the **characters** after conversion to text.

void operator += (const long *number* **);**
Appends *number* to the **characters** after conversion to text.

operator char * ( ) const ;
Returns the address of the first byte in the **storage**. This value can be zero, and is volatile. Use this method only for read-only purposes.

ImqBoolean operator < (const ImqString & *string* **) const ;**
Compares the **characters** with those of *string* using the **compare** method. The result is TRUE if less than and FALSE if greater than or equal to.

ImqBoolean operator > (const ImqString & *string* **) const ;**
Compares the **characters** with those of *string* using the **compare** method. The result is TRUE if greater than and FALSE if less than or equal to.

ImqBoolean operator <= (const ImqString & *string* **) const ;**
Compares the **characters** with those of *string* using the **compare** method. The result is TRUE if less than or equal to and FALSE if greater than.

ImqBoolean operator >= (const ImqString & *string* **) const ;**
Compares the **characters** with those of *string* using the **compare** method. The result is TRUE if greater than or equal to and FALSE if less than.

ImqBoolean operator == (const ImqString & *string* **) const ;**

Compares the **characters** with those of *string* using the **compare** method. It returns either TRUE or FALSE.

ImqBoolean operator != (const ImqString & *string* **) const ;**

Compares the **characters** with those of *string* using the **compare** method. It returns either TRUE or FALSE.

short compare(const ImqString & *string* **) const ;**

Compares the **characters** with those of *string*. The result is zero if the **characters** are equal, negative if less than and positive if greater than. Comparison is case sensitive. A null ImqString is regarded as less than a nonnull ImqString.

ImqBoolean copyOut(char * *buffer*, **const size_t** *length*, **const char** *pad* **= 0);**

Copies up to *length* bytes from the **characters** to the *buffer*. If the number of **characters** is insufficient, fills the remaining space in *buffer* with *pad* characters. *buffer* can be zero if *length* is also zero. It returns TRUE if successful.

size_t copyOut(long & *number* **) const ;**

Sets *number* from the **characters** after conversion from text, and returns the number of characters involved in the conversion. If this is zero, no conversion has been performed and *number* is not set. A convertible character sequence must begin with the following values:

```
<blank(s)>
<+|->
digit(s)
```

size_t copyOut(ImqString & *token*, **const char** *c* **= ' ') const ;**

If the **characters** contain one or more characters that are different from *c*, identifies a token as the first contiguous sequence of such characters. In this case *token* is set to that sequence, and the value returned is the sum of the number of leading characters *c* and the number of bytes in the sequence. Otherwise, returns zero and does not set *token*.

size_t cutOut(long & *number* **);**

Sets *number* as for the **copy** method, but also removes from **characters** the number of bytes indicated by the return value. For example, the string shown in the following example can be cut into three numbers by using **cutOut**(*number*) three times:

```
strNumbers = "-1 0     +55 "

while ( strNumbers.cutOut( number ) );
number becomes -1, then 0, then 55
leaving strNumbers == " "
```

size_t cutOut(ImqString & *token*, **const char** *c* **= ' ' '**

Sets *token* as for the **copyOut** method, and removes from **characters** the *strToken* characters and also any characters *c* that precede the *token* characters. If *c* is not a blank, removes characters *c* that directly succeed the *token* characters. Returns the number of characters removed. For example, the string shown in the following example can be cut into three tokens by using **cutOut**(*token*) three times:

```
strText = "  Program Version 1.1  "

while ( strText.cutOut( token ) );

// token becomes "Program", then "Version",
// then "1.1" leaving strText == "  "
```

The following example shows how to parse a DOS path name:

```
strPath = "C:\OS2\BITMAP\OS2LOGO.BMP"

strPath.cutOut( strDrive, ':' );
strPath.stripLeading( ':' );
while ( strPath.cutOut( strFile, '\' ) );
```

```
// strDrive becomes "C".
// strFile becomes "OS2", then "BITMAP",
// then "OS2LOGO.BMP" leaving strPath empty.
```

ImqBoolean find(const ImqString & *string* **);**
Searches for an exact match for *string* anywhere within the **characters**. If no match is found, it returns FALSE. Otherwise, it returns TRUE. If *string* is null, it returns TRUE.

ImqBoolean find(const ImqString & *string*, **size_t &** *offset* **);**
Searches for an exact match for *string* somewhere within the **characters** from offset *offset* onwards. If *string* is null, it returns TRUE without updating *offset*. If no match is found, it returns FALSE (the value of *offset* might have been increased). If a match is found, it returns TRUE and updates *offset* to the offset of *string* within the **characters**.

size_t length( ) const ;
Returns the **length**.

ImqBoolean pasteIn(const double *number*, **const char *** *format* **= "%f");**
Appends *number* to the **characters** after conversion to text. It returns TRUE if successful.

The specification *format* is used to format the floating point conversion. If specified, it must be one suitable for use with **printf** and floating point numbers, for example **%.3f**.

ImqBoolean pasteIn(const long *number* **);**
Appends *number* to the **characters** after conversion to text. It returns TRUE if successful.

ImqBoolean pasteIn(const void * *buffer*, **const size_t** *length* **);**
Appends *length* bytes from *buffer* to the **characters**, and adds a final trailing null. Substitutes any null characters copied. The substitution character is a period (.). No special consideration is given to any other nonprintable or nondisplayable characters copied. This method returns TRUE if successful.

ImqBoolean set(const char * *buffer*, **const size_t** *length* **);**
Sets the **characters** from a fixed-length character field, which might contain a null. Appends a null to the characters from the fixed-length field if necessary. This method returns TRUE if successful.

ImqBoolean setStorage(const size_t *length* **);**
Allocates (or reallocates) the **storage**. Preserves any original **characters**, including any trailing null, if there is still room for them, but does not initialize any additional storage.

This method returns TRUE if successful.

size_t storage( ) const ;
Returns the number of bytes in the **storage**.

size_t stripLeading(const char *c* **= ' ');**
Strips leading characters *c* from the **characters** and returns the number removed.

size_t stripTrailing(const char *c* **= ' ');**
Strips trailing characters *c* from the **characters** and returns the number removed.

ImqString upperCase( ) const ;
Returns an uppercase copy of the **characters**.

Object methods (protected)

ImqBoolean assign(const ImqString & *string* **);**
Equivalent to the equivalent **operator =** method, but non-virtual. It returns TRUE if successful.

Reason codes

- MQRC_DATA_TRUNCATED
- MQRC_NULL_POINTER

- MQRC_STORAGE_NOT_AVAILABLE
- MQRC_BUFFER_ERROR
- MQRC_INCONSISTENT_FORMAT

ImqTrigger C++ class

This class encapsulates the MQTM (trigger message) data structure.

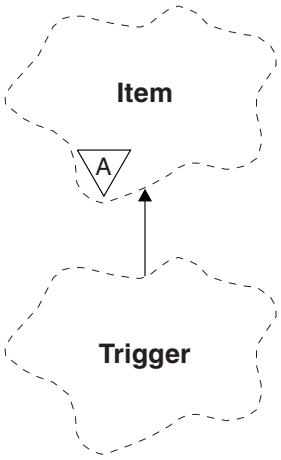


Figure 77. ImqTrigger class

Objects of this class are typically used by a trigger monitor program. The task of a trigger monitor program is to wait for these particular messages and act on them to ensure that other WebSphere MQ applications are started when messages are waiting for them.

See the IMQSTRG sample program for a usage example.

- "Object attributes"
- "Constructors" on page 2745
- "Overloaded ImqItem methods" on page 2745
- "Object methods (public)" on page 2745
- "Object data (protected)" on page 2746
- "Reason codes" on page 2746

Object attributes

application id
Identity of the application that sent the message. The initial value is a null string.

application type
Type of application that sent the message. The initial value is zero. The following additional values are possible:

- MQAT_AIX
- MQAT_CICS
- MQAT_DOS
- MQAT_IMS
- MQAT_MVS
- MQAT_NOTES_AGENT
- MQAT_OS2
- MQAT_OS390

- MQAT_OS400
- MQAT_UNIX
- MQAT_WINDOWS
- MQAT_WINDOWS_NT
- MQAT_USER_FIRST
- MQAT_USER_LAST

environment data
Environment data for the process. The initial value is a null string.

process name
Process name. The initial value is a null string.

queue name
Name of the queue to be started. The initial value is a null string.

trigger data
Trigger data for the process. The initial value is a null string.

user data
User data for the process. The initial value is a null string.

Constructors

ImqTrigger( );
The default constructor.

ImqTrigger(const ImqTrigger & *trigger* **);**
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg* **);**
Writes an MQTM data structure to the message buffer, replacing any existing content. Sets the *msg* **format** to MQFMT_TRIGGER.

See the ImqItem class method description at "ImqItem C++ class" on page 2686 for further details.

virtual ImqBoolean pasteIn(ImqMessage & *msg* **);**
Reads an MQTM data structure from the message buffer.

To be successful, the ImqMessage **format** must be MQFMT_TRIGGER.

See the ImqItem class method description at "ImqItem C++ class" on page 2686 for further details.

Object methods (public)

void operator = (const ImqTrigger & *trigger* **);**
Copies instance data from *trigger*, replacing the existing instance data.

ImqString applicationId( ) const ;
Returns a copy of the **application id**.

void setApplicationId(const char * *id* **);**
Sets the **application id**.

MQLONG applicationType( ) const ;
Returns the **application type**.

void setApplicationType(const MQLONG *type* **);**
Sets the **application type**.

ImqBoolean copyOut(MQTMC2 * *ptmc2*);
Encapsulates the MQTM data structure, which is the one received on initiation queues. Fills in an equivalent MQTMC2 data structure provided by the caller, and sets the QMgrName field (which is not present in the MQTM data structure) to all blanks. The MQTMC2 data structure is traditionally used as a parameter to applications started by a trigger monitor. This method returns TRUE if successful.

ImqString environmentData( ) const ;
Returns a copy of the **environment data**.

void setEnvironmentData(const char * *data*);
Sets the **environment data**.

ImqString processName( ) const ;
Returns a copy of the **process name**.

void setProcessName(const char * *name*);
Sets the **process name**, padded with blanks to 48 characters.

ImqString queueName( ) const ;
Returns a copy of the **queue name**.

void setQueueName(const char * *name*);
Sets the **queue name**, padding with blanks to 48 characters.

ImqString triggerData( ) const ;
Returns a copy of the **trigger data**.

void setTriggerData(const char * *data*);
Sets the **trigger data**.

ImqString userData( ) const ;
Returns a copy of the **user data**.

void setUserData(const char * *data*);
Sets the **user data**.

Object data (protected)

MQTM *omqtm*
The MQTM data structure.

Reason codes

- MQRC_NULL_POINTER
- MQRC_INCONSISTENT_FORMAT
- MQRC_ENCODING_ERROR
- MQRC_STRUC_ID_ERROR

ImqWorkHeader C++ class

This class encapsulates specific features of the MQWIH data structure.

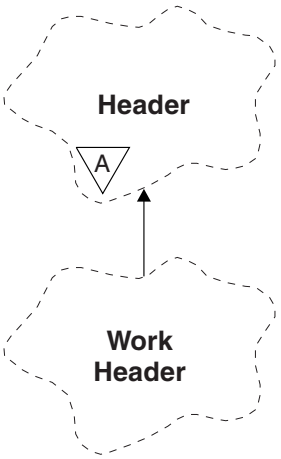


Figure 78. ImqWorkHeader class

Objects of this class are used by applications putting messages to the queue managed by the z/OS Workload Manager.

Object attributes

message token
Message token for the z/OS Workload Manager, of length MQ_MSG_TOKEN_LENGTH. The initial value is MQMTOK_NONE.

service name
The 32-character name of a process. The name is initially blanks.

service step
The 8-character name of a step within the process. The name is initially blanks.

Constructors

ImqWorkHeader( );
The default constructor.

ImqWorkHeader(const ImqWorkHeader & *header* **);**
The copy constructor.

Overloaded ImqItem methods

virtual ImqBoolean copyOut(ImqMessage & *msg* **);**
Inserts an MQWIH data structure into the beginning of the message buffer, moving the existing message data further along, and sets the *msg* **format** to MQFMT_WORK_INFO_HEADER.

See the parent class method description for more details.

virtual ImqBoolean pasteIn(ImqMessage & *msg* **);**
Reads an MQWIH data structure from the message buffer.

To be successful, the encoding of the *msg* object must be MQENC_NATIVE. Retrieve messages with MQGMO_CONVERT to MQENC_NATIVE.

The ImqMessage format must be MQFMT_WORK_INFO_HEADER.

See the parent class method description for more details.

Object methods (public)

void operator = (const ImqWorkHeader & *header* **);**
Copies instance data from *header*, replacing the existing instance data.

ImqBinary messageToken ( ) const;
Returns the **message token**.

ImqBoolean setMessageToken(const ImqBinary & *token* **);**
Sets the **message token**. The data length of *token* must be either zero or MQ_MSG_TOKEN_LENGTH. It returns TRUE if successful.

void setMessageToken(const MQBYTE16 *token* **= 0);**
Sets the **message token**. *token* can be zero, which is the same as specifying MQMTOK_NONE. If *token* is nonzero, it must address MQ_MSG_TOKEN_LENGTH bytes of binary data.

When using predefined values such as MQMTOK_NONE, you might need make a cast to ensure a signature match; for example, (MQBYTE *)MQMTOK_NONE.

ImqString serviceName ( ) const;
Returns the **service name**, including trailing blanks.

void setServiceName(const char * *name* **);**
Sets the **service name**.

ImqString serviceStep ( ) const;
Returns the **service step**, including trailing blanks.

void setServiceStep(const char * *step* **);**
Sets the **service step**.

Object data (protected)

MQWIH omqwih
The MQWIH data structure.

Reason codes

- MQRC_BINARY_DATA_LENGTH_ERROR

The WebSphere MQ classes for Java libraries

The location of the WebSphere MQ classes for Java libraries varies according to platform. Specify this location when you start an application.

To specify the location of the Java Native Interface (JNI) libraries, start your application using a **java** command with the following format:

```
java -Djava.library.path=library_path application_name
```

where *library_path* is the path to the WebSphere MQ classes for Java libraries, which include the JNI libraries. Table 262 on page 2749 shows the location of the WebSphere MQ classes for Java libraries for each platform.

Table 262. The location of the WebSphere MQ classes for Java libraries for each platform

Platform	Directory containing the WebSphere MQ classes for Java libraries
AIX	*MQ_INSTALLATION_PATH*/java/lib (32 bit libraries) *MQ_INSTALLATION_PATH*/java/lib64 (64 bit libraries)
HP-UX Linux (POWER, x86-64 and zSeries s390x platforms) Solaris (x86-64 and SPARC platforms)	*MQ_INSTALLATION_PATH*/java/lib (32 bit libraries) *MQ_INSTALLATION_PATH*/java/lib64 (64 bit libraries)
Linux (x86 platform)	*MQ_INSTALLATION_PATH*/java/lib
Windows	*MQ_INSTALLATION_PATH*\Java\lib (32 bit libraries) *MQ_INSTALLATION_PATH*\Java\lib64 (64 bit libraries)
MQ_INSTALLATION_PATH represents the high-level directory in which WebSphere MQ is installed.	

Note:

1. On AIX, HP-UX, Linux (Power platform), or Solaris, use either the 32 bit libraries or the 64 bit libraries. Use the 64 bit libraries only if you are running your application in a 64 bit Java virtual machine (JVM) on a 64 bit platform. Otherwise, use the 32 bit libraries.
2. On Windows, you can use the PATH environment variable to specify the location of the WebSphere MQ classes for Java libraries instead of specifying their location on the **java** command.
3. To use WebSphere MQ classes for Java in bindings mode on IBM i, ensure that the library QMQMJAVA is in your library list.

Related information:

Using WebSphere MQ classes for Java

Properties of IBM WebSphere MQ classes for JMS objects

All objects in IBM WebSphere MQ classes for JMS have properties. Different properties apply to different object types. Different properties have different allowable values, and symbolic property values differ between the administration tool and program code.

IBM WebSphere MQ classes for JMS provides facilities to set and query the properties of objects using the WebSphere MQ JMS administration tool, WebSphere MQ Explorer, or in an application. Many of the properties are relevant only to a specific subset of the object types.

For information on how you use the WebSphere MQ JMS administration tool, see Using the WebSphere MQ JMS administration tool .

Table 263 on page 2750 gives a brief description of each property and shows for each property which object types it applies to. The object types are identified using keywords; see JMS object types for an explanation of these.

Numbers refer to notes at the end of the table. See also “Dependencies between properties of WebSphere MQ classes for JMS objects” on page 2801.

A property consists of a name-value pair in the format:

```
PROPERTY_NAME(property_value)
```

The topics in this section list, for each property, the name of the property and a brief description, and shows the valid property values used in the administration tool. and the set method that is used to set the value of the property in an application. The topics also show the valid property values for each property and the mapping between symbolic property values used in the tool and their programmable equivalents.

Property names are not case-sensitive, and are restricted to the set of recognized names shown in these topics.

Table 263. Property names and applicable object types

Property	Short form	Object type							
		CF	QCF	TCF	Q	T	XACF	XAQCF	XATCF
"APPLICATIONNAME" on page 2753	APPNAME	Y	Y	Y			Y	Y	Y
"ASYNCEXCEPTION" on page 2753	AEX	Y	Y	Y			Y	Y	Y
"BROKERCCDURSUBQ" on page 2754 [1]	CCDSUB					Y			
"BROKERCCSUBQ" on page 2755 [1]	CCSUB	Y		Y			Y		Y
"BROKERCONQ" on page 2755 [1]	BCON	Y		Y			Y		Y
"BROKERDURSUBQ" on page 2756 [1]	BDSUB					Y			
"BROKERPUBQ" on page 2756 [1]	BPUB	Y		Y		Y	Y		Y
"BROKERPUBQMGR" on page 2757 [1]	BPQM					Y			
"BROKERQMGR" on page 2757 [1]	BQM	Y		Y			Y		Y
"BROKERSUBQ" on page 2758 [1]	BSUB	Y		Y			Y		Y
"BROKERVER" on page 2758 [1]	BVER	Y[2]		Y[2]		Y	Y		Y
"CCDTURL" on page 2759 [3]	CCDT	Y	Y	Y			Y	Y	Y
"CCSID" on page 2759	CCS	Y	Y	Y	Y	Y	Y	Y	Y
"CHANNEL" on page 2760 [3]	CHAN	Y	Y	Y			Y	Y	Y
"CLEANUP" on page 2760 [1]	CL	Y		Y			Y		Y
"CLEANUPINT" on page 2761 [1]	CLINT	Y		Y			Y		Y
"CONNECTIONNAMELIST" on page 2761	CNLIST	Y	Y	Y					
"CLIENTRECONNECTOPTIONS" on page 2762	CROPT	Y	Y	Y					
"CLIENTRECONNECTTIMEOUT" on page 2763	CRT	Y	Y	Y					
"CLIENTID" on page 2763	CID	Y[2]	Y	Y[2]			Y	Y	Y
"CLONESUPP" on page 2764	CLS	Y		Y			Y		Y
"COMPHDR" on page 2764	HC	Y		Y			Y		Y
"COMPMSG" on page 2765	MC	Y	Y	Y			Y	Y	Y
"CONNOPT" on page 2765	CNOPT	Y	Y	Y			Y	Y	Y
"CONNTAG" on page 2766	CNTAG	Y	Y	Y			Y	Y	Y
"DESCRIPTION" on page 2767	DESC	Y[2]	Y	Y[2]	Y	Y	Y	Y	Y
"DIRECTAUTH" on page 2767	DAUTH	Y[2]		Y[2]					
"ENCODING" on page 2768	ENC				Y	Y			
"EXPIRY" on page 2769	EXP				Y	Y			
"FAILIFQUIESCE" on page 2769	FIQ	Y	Y	Y	Y	Y	Y	Y	Y
"HOSTNAME" on page 2770	HOST	Y[2]	Y	Y[2]			Y	Y	Y
"LOCALADDRESS" on page 2770	LA	Y[2]	Y	Y[2]			Y	Y	Y

Table 263. Property names and applicable object types (continued)

Property	Short form	Object type							
		CF	QCF	TCF	Q	T	XACF	XAQCF	XATCF
"MAPNAMESTYLE" on page 2771	MNST	Y	Y	Y			Y	Y	Y
"MAXBUFFSIZE" on page 2772	MBSZ	Y[2]		Y[2]					
"MDREAD" on page 2772	MDR				Y	Y			
"MDWRITE" on page 2773	MDW				Y	Y			
"MDMSGCTX" on page 2773	MDCTX				Y	Y			
"MSGBATCHSZ" on page 2774 [1]	MBS	Y	Y	Y			Y	Y	Y
"MSGBODY" on page 2774	MBODY				Y	Y			
"MSGRETENTION" on page 2775	MRET	Y	Y				Y	Y	
"MSGSELECTION" on page 2775 [1]	MSEL	Y		Y			Y		Y
"MULTICAST" on page 2776	MCAST	Y[2]		Y[2]		Y			
"OPTIMISTICPUBLICATION" on page 2777 [1]	OPTPUB	Y		Y					
"OUTCOMENOTIFICATION" on page 2777 [1]	NOTIFY	Y		Y					
"PERSISTENCE" on page 2778	PER				Y	Y			
"POLLINGINT" on page 2778 [1]	PINT	Y	Y	Y			Y	Y	Y
"PORT" on page 2779	PORT	Y[2]	Y	Y[2]			Y	Y	Y
"PRIORITY" on page 2779	PRI				Y	Y			
"PROCESSDURATION" on page 2780 [1]	PROCDUR	Y		Y					
"PROVIDERVERSION" on page 2780	PVER	Y	Y	Y			Y	Y	Y
"PROXYHOSTNAME" on page 2782	PHOST	Y[2]		Y[2]					
"PROXYPORT" on page 2782	PPORT	Y[2]		Y[2]					
"PUBACKINT" on page 2783 [1]	PAI	Y		Y			Y		Y
"PUTASYNCALLOWED" on page 2783	PAALD				Y	Y			
"QMANAGER" on page 2784	QMGR	Y	Y	Y	Y		Y	Y	Y
"QUEUE" on page 2784	QU				Y				
"READAHEADALLOWED" on page 2785	RAALD				Y	Y			
"READAHEADCLOSEPOLICY" on page 2785	RACP				Y	Y			
"RECEIVECCSID" on page 2786	RCCS				Y	Y			
"RECEIVECONVERSION" on page 2786	RCNV				Y	Y			
"RECEIVEISOLATION" on page 2787 [1]	RCVISOL	Y		Y					
"RECEXIT" on page 2787	RCX	Y	Y	Y			Y	Y	Y
"RECEXITINIT" on page 2788	RCXI	Y	Y	Y			Y	Y	Y
"REPLYTOSTYLE" on page 2788	RTOST				Y	Y			
"RESCANINT" on page 2789 [1]	RINT	Y	Y				Y	Y	

Table 263. Property names and applicable object types (continued)

Property	Short form	Object type							
		CF	QCF	TCF	Q	T	XACF	XAQCF	XATCF
"SECEXIT" on page 2790	SCX	Y	Y	Y			Y	Y	Y
"SECEXITINIT" on page 2790	SCXI	Y	Y	Y			Y	Y	Y
"SENDCHECKCOUNT" on page 2791	SCC	Y	Y	Y			Y	Y	Y
"SENDEXIT" on page 2791	SDX	Y	Y	Y			Y	Y	Y
"SENDEXITINIT" on page 2792	SDXI	Y	Y	Y			Y	Y	Y
"SHARECONVALLOWED" on page 2792	SCALD	Y	Y	Y			Y	Y	Y
"SPARSESUBS" on page 2793 [1]	SSUBS	Y		Y					
"SSLCIPHERSUITE" on page 2793	SCPHS	Y	Y	Y			Y	Y	Y
"SSLCRL" on page 2794	SCRL	Y	Y	Y			Y	Y	Y
"SSLFIPSREQUIRED" on page 2794	SFIPS	Y	Y	Y			Y	Y	Y
"SSLPEERNAME" on page 2795	SPEER	Y	Y	Y			Y	Y	Y
"SSLRESETCOUNT" on page 2795	SRC	Y	Y	Y			Y	Y	Y
"STATREFRESHINT" on page 2796 [1]	SRI	Y		Y			Y		Y
"SUBSTORE" on page 2796 [1]	SS	Y		Y			Y		Y
"SYNCPOINTALLGETS" on page 2797	SPAG	Y	Y	Y			Y	Y	Y
"TARGCLIENT" on page 2797	TC				Y	Y			
"TARGCLIENTMATCHING" on page 2798	TCM	Y	Y				Y	Y	
"TEMPMODEL" on page 2798	TM	Y	Y				Y	Y	
"TEMPQPREFIX" on page 2799	TQP	Y	Y				Y	Y	
"TEMPTOPICPREFIX" on page 2799	TTP	Y		Y			Y		Y
"TOPIC" on page 2800	TOP					Y			
"TRANSPORT" on page 2800	TRAN	Y[2]	Y	Y[2]			Y	Y	Y
"WILDCARDFORMAT" on page 2801	WCFMT	Y		Y			Y		Y

Note:

1. This property can be used with Version 7.0 of WebSphere MQ classes for JMS but has no effect for an application connected to a Version 7.0 queue manager unless the PROVIDERVERSION property of the connection factory is set to a version number less than 7.
2. Only the BROKERVER, CLIENTID, DESCRIPTION, DIRECTAUTH, HOSTNAME, LOCALADDRESS, MAXBUFFSIZE, MULTICAST, PORT, PROXYHOSTNAME, PROXYPORT, and TRANSPORT properties are supported for a ConnectionFactory or TopicConnectionFactory object when using a real-time connection to a broker.
3. The CCDTURL and CHANNEL properties of an object must not both be set at the same time.

APPLICATIONNAME

An application can set a name that identifies its connection to the queue manager. This application name is shown by the **DISPLAY CONN MQSC/PCF** command (where the field is called **APPLTAG**) or in the WebSphere MQ Explorer Application Connections display (where the field is called **App name**).

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: APPLICATIONNAME

JMS administration tool short name: APPNAME

Programmatic access

Setters/getters

- MQConnectionFactory.setAppName()
- MQConnectionFactory.getAppName()

Values

Any valid string that is no longer than 28 characters. Longer names are adjusted to fit by removing leading package names, if necessary. For example, if the invoking class is `com.example.MainApp`, the full name is used, but if the invoking class is `com.example.dictionaryAndThesaurus.multilingual.mainApp`, the name `multilingual.mainApp` is used, because it is the longest combination of class name and rightmost package name that fits into the available length.

If the class name itself is more than 28 characters long, it is truncated to fit. For example, `com.example.mainApplicationForSecondTestCase` becomes `mainApplicationForSecondTest`.

ASYNCEXCEPTION

This property determines whether WebSphere MQ classes for JMS informs an ExceptionListener only when a connection is broken, or when any exception occurs asynchronously to a JMS API call. This applies to all Connections created from this ConnectionFactory that have an ExceptionListener registered.

Applicable objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: ASYNCEXCEPTION

JMS administration tool short name: AEX

Programmatic access

Setters/Getters

- MQConnectionFactory.setAsyncExceptions()
- MQConnectionFactory.getAsyncExceptions()

Values

ASYNC_EXCEPTIONS_ALL

Any exception detected asynchronously, outside the scope of a synchronous API call, and all connection broken exceptions are sent to the ExceptionListener.

Environment	Value
JMS Administration Tool	ASYNC_EXCEPTIONS_ALL
Programmatic	WMQCONSTANTS.ASYNC_EXCEPTIONS_ALL = -1
WebSphere MQ Explorer	All

ASYNC_EXCEPTIONS_CONNECTIONBROKEN

Only exceptions indicating a broken connection are sent to the ExceptionListener. Any other exceptions occurring during asynchronous processing are not reported to the ExceptionListener, and hence the application is not informed of these exceptions. This is the default value from IBM WebSphere MQ Version 7.5.0, Fix Pack 8 (see JMS: Exception listener changes in Version 7.5).

Environment	Value
JMS Administration Tool	ASYNC_EXCEPTIONS_CONNECTIONBROKEN
Programmatic	WMQCONSTANTS.ASYNC_EXCEPTIONS_CONNECTIONBROKEN = 1
WebSphere MQ Explorer	Connection Broken

The following additional constant is defined:

- From Version 7.5.0, Fix Pack 8: WMQCONSTANTS.ASYNC_EXCEPTIONS_DEFAULT = ASYNC_EXCEPTIONS_CONNECTIONBROKEN
- Before Version 7.5.0, Fix Pack 8: WMQCONSTANTS.ASYNC_EXCEPTIONS_DEFAULT = ASYNC_EXCEPTIONS_ALL

Related information:

Exceptions in WebSphere MQ classes for JMS

BROKERCCDURSUBQ

The name of the queue from which durable subscription messages are retrieved for a ConnectionConsumer.

Applicable objects

Topic

JMS administration tool long name: BROKERCCDURSUBQ

JMS administration tool short name: CCDSUB

Programmatic access

Setters/getters

- MQTopic.setBrokerCCDurSubQueue()
- MQTopic.getBrokerCCDurSubQueue()

Values

SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
This is the default value.

Any valid string

BROKERCCSUBQ

The name of the queue from which non-durable subscription messages are retrieved for a ConnectionConsumer.

Applicable objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERCCSUBQ

JMS administration tool short name: CCSUB

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerCCSubQueue()
- MQConnectionFactory.getBrokerCCSubQueue()

Values

SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
This is the default value.

Any valid string

BROKERCONQ

The control queue name of the broker.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERCONQ

JMS administration tool short name: BCON

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerControlQueue()
- MQConnectionFactory.getBrokerControlQueue()

Values

SYSTEM.BROKER.CONTROL.QUEUE
This is the default value.

Any valid string

BROKERDURSUBQ

When the WebSphere MQ classes for JMS are being used in WebSphere MQ messaging provider migration mode, this property specifies the name of the queue from which durable subscription messages are retrieved.

Applicable objects

Topic

JMS administration tool long name: BROKERDURSUBQ

JMS administration tool short name: BDSUB

Programmatic access

Setters/getters

- MQTopic.setBrokerDurSubQueue()
- MQTopic.getBrokerDurSubQueue()

Values

SYSTEM.JMS.D.SUBSCRIBER.QUEUE
This is the default value.

Any valid string
Starting with SYSTEM.JMS.D

Related information:

Rules for selecting the WebSphere MQ messaging provider mode

BROKERPUBQ

The name of the queue where published messages are sent (the stream queue).

Applicable Objects

ConnectionFactory, TopicConnectionFactory, Topic, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERPUBQ

JMS administration tool short name: BPUB

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerPubQueue
- MQConnectionFactory.getBrokerPubQueue

Values

SYSTEM.BROKER.DEFAULT.STREAM
This is the default value.

Any valid string

BROKERPUBQMGR

The name of the queue manager that owns the queue where messages published on the topic are sent.

Applicable Objects

Topic

JMS administration tool long name: BROKERPUBQMGR

JMS administration tool short name: BPQM

Programmatic access

Setters/getters

- MQTopic.setBrokerPubQueueManager()
- MQTopic.getBrokerPubQueueManager()

Values

null This is the default value.

Any valid string

BROKERQMGR

The name of the queue manager on which the broker is running.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERQMGR

JMS administration tool short name: BQM

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerQueueManager()
- MQConnectionFactory.getBrokerQueueManager()

Values

null This is the default value.

Any valid string

BROKERSUBQ

When the WebSphere MQ classes for JMS are being used in WebSphere MQ messaging provider migration mode, this property specifies the name of the queue from which non-durable subscription messages are retrieved.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERSUBQ

JMS administration tool short name: BSUB

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerSubQueue()
- MQConnectionFactory.getBrokerSubQueue()

Values

SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
This is the default value.

Any valid string
Starting with SYSTEM.JMS.ND

Related information:

Rules for selecting the WebSphere MQ messaging provider mode

BROKERVER

The version of the broker being used.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, Topic, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: BROKERVER

JMS administration tool short name: BVER

Programmatic access

Setters/getters

- MQConnectionFactory.setBrokerVersion()
- MQConnectionFactory.getBrokerVersion()

Values

V1 To use a WebSphere MQ Publish/Subscribe broker, or to use a broker of WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker in compatibility mode. This is the default value if TRANSPORT is set to BIND or CLIENT.

V2 To use a broker of WebSphere MQ Integrator, WebSphere Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker in native mode. This is the default value if TRANSPORT is set to DIRECT or DIRECTHTTP.

unspecified
After the broker has migrated from V6 to V7, set this property so that RFH2 headers are no longer used. After migration this property is no longer relevant.

CCDTURL

A Uniform Resource Locator (URL) that identifies the name and location of the file containing the client channel definition table and specifies how the file can be accessed.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CCDTURL

JMS administration tool short name: CCDT

Programmatic access

Setters/getters

- MQConnectionFactory.setCCDTURL()
- MQConnectionFactory.getCCDTURL()

Values

null This is the default value.

A Uniform Resource Locator (URL)

CCSID

The coded character set ID to be used for a connection or destination.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, Queue, Topic, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CCSID

JMS administration tool short name: CCS

Programmatic access

Setters/getters

- MQConnectionFactory.setCCSID()
- MQConnectionFactory.getCCSID()

Values

819 This is the default value for a connection factory.

1208 This is the default value for a destination.

Any positive integer

CHANNEL

The name of the client connection channel being used.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CHANNEL

JMS administration tool short name: CHAN

Programmatic access

Setters/getters

- MQConnectionFactory.setChannel()
- MQConnectionFactory.getChannel()

Values

SYSTEM.DEF.SVRCONN
This is the default value.

Any valid string

CLEANUP

Cleanup Level for BROKER or MIGRATE Subscription Stores.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CLEANUP

JMS administration tool short name: CL

Programmatic access

Setters/getters

- MQConnectionFactory.setCleanupLevel()
- MQConnectionFactory.getCleanupLevel()

Values

SAFE Use safe cleanup. This is the default value.

ASPROP
Use safe, strong, or no cleanup according to a property set on the Java command line.

NONE
Use no cleanup.

STRONG
Use strong cleanup.

CLEANUPINT

The interval, in milliseconds, between background executions of the publish/subscribe cleanup utility.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CLEANUPINT

JMS administration tool short name: CLINT

Programmatic access

Setters/getters

- MQConnectionFactory.setCleanupInterval()
- MQConnectionFactory.getCleanupInterval()

Values

3600000
This is the default value.

Any positive integer

CONNECTIONNAMELIST

List of TCP/IP connection names. The list is tried in order, once per each reconnection retry attempt.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory

JMS administration tool long name: CONNECTIONNAMELIST

JMS administration tool short name: CNLIST

Programmatic access

Setters/getters

- MQConnectionFactory.setconnectionNameList()
- MQConnectionFactory.getconnectionNameList()

Values

Comma separated list of HOSTNAME(PORT). HOSTNAME(PORT) can be either a DNS name or IP address.

PORT defaults to 1414.

CLIENTRECONNECTOPTIONS

Options governing reconnection.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory

JMS administration tool long name: CLIENTRECONNECTOPTIONS

JMS administration tool short name: CROPT

Programmatic access

Setters/getters

- MQConnectionFactory.setClientReconnectOptions()
- MQConnectionFactory.getClientReconnectOptions()

Values

QMGR

The application can reconnect, but only to the queue manager to which it originally connected.

Use this value if an application can be reconnected, but there is an affinity between the WebSphere MQ classes for JMS application, and the queue manager to which it first established a connection.

Choose this value if you want an application to automatically reconnect to the standby instance of a highly available queue manager.

To use this value programmatically, use the constant WMQConstants.WMQ_CLIENT_RECONNECT_Q_MGR.

ANY

The application can reconnect to any queue manager.

Use the reconnect option only if there is no affinity between the WebSphere MQ classes for JMS application and the queue manager with which it initially established a connection.

To use this value from a program, use the constant WMQConstants.WMQ_CLIENT_RECONNECT.

DISABLED

The application will not be reconnected.

To use this value programmatically, use the constant WMQConstants.WMQ_CLIENT_RECONNECT_DISABLED.

ASDEF

Whether the application will reconnect automatically depends on the value of the WebSphere MQ channel attribute DefReconnect.

To use this value from a program, use the constant WMQConstants.WMQ_CLIENT_RECONNECT_AS_DEF.

CLIENTRECONNECTTIMEOUT

Time before reconnection retries cease.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory

JMS administration tool long name: CLIENTRECONNECTTIMEOUT

JMS administration tool short name: CRT

Programmatic access

Setters/getters

- MQConnectionFactory.setClientReconnectTimeout()
- MQConnectionFactory.setClientReconnectTimeout()

Values

Interval in seconds. Default 1800 (30 minutes).

CLIENTID

The client identifier is used to uniquely identify the application connection for durable subscriptions.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CLIENTID

JMS administration tool short name: CID

Programmatic access

Setters/getters

- MQConnectionFactory.setClientId()
- MQConnectionFactory.getClientId()

Values

null This is the default value.

Any valid string

CLONESUPP

Whether two or more instances of the same durable topic subscriber can run simultaneously.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CLONESUPP

JMS administration tool short name: CLS

Programmatic access

Setters/getters

- MQConnectionFactory.setCloneSupport()
- MQConnectionFactory.getCloneSupport()

Values

DISABLED
Only one instance of a durable topic subscriber can run at a time. This is the default value.

ENABLED
Two or more instances of the same durable topic subscriber can run simultaneously, but each instance must run in a separate Java virtual machine (JVM).

COMPHDR

A list of the techniques that can be used for compressing header data on a connection.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: COMPHDR

JMS administration tool short name: HC

Programmatic access

Setters/getters

- MQConnectionFactory.setHdrCompList()
- MQConnectionFactory.getHdrCompList()

Values

NONE
This is the default value.

SYSTEM
RLE message header compression is performed.

COMPMSG

A list of the techniques that can be used for compressing message data on a connection.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: COMPMSG

JMS administration tool short name: MC

Programmatic access

Setters/getters

- MQConnectionFactory.setMsgCompList()
- MQConnectionFactory.getMsgCompList()

Values

NONE
This is the default value.

A list of one or more of the following values separated by blank characters:
RLE ZLIBFAST ZLIBHIGH

CONNOPT

Controls how WebSphere MQ classes for JMS applications that use the bindings transport connect to the queue manager.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CONNOPT

JMS administration tool short name: CNOPT

Programmatic access

Setters/getters

- MQConnectionFactory.setMQConnectionOptions()
- MQConnectionFactory.getMQConnectionOptions()

Values

STANDARD
The nature of the binding between the application and the queue manager depends on the value of the *DefaultBindType* attribute of the queue manager, The STANDARD value maps to the WebSphere MQ *ConnectOption* MQCNO_STANDARD_BINDING

SHARED
The application and the local queue manager agent run in separate units of execution but share some resources. This value maps to the WebSphere MQ *ConnectOption* MQCNO_SHARED_BINDING.

ISOLATED
The application and the local queue manager agent run in separate units of execution and share no resources. The ISOLATED value maps to the WebSphere MQ *ConnectOption* MQCNO_ISOLATED_BINDING.

FASTPATH
The application and the local queue manager agent run in the same unit of execution. This value maps to the WebSphere MQ *ConnectOption* MQCNO_FASTPATH_BINDING .

SERIALQM
The application requests exclusive use of the connection tag within the scope of the queue manager. This value maps to the WebSphere MQ *ConnectOption* MQCNO_SERIALIZE_CONN_TAG_Q_MGR.

SERIALQSG
The application requests exclusive use of the connection tag within the scope of the queue sharing group to which the queue manager belongs. The SERIALQSG value maps to the WebSphere MQ *ConnectOption* MQCNO_SERIALIZE_CONN_TAG_QSG.

RESTRICTQM
The application requests shared use of the connection tag, but there are restrictions on the shared use of the connection tag within the scope of the queue manager. This value maps to the WebSphere MQ *ConnectOption* MQCNO_RESTRICT_CONN_TAG_Q_MGR.

RESTRICTQSG
The application requests shared use of the connection tag, but there are restrictions on the shared use of the connection tag within the scope of the queue sharing group to which the queue manager belongs. This value maps to the WebSphere MQ *ConnectOption* MQCNO_RESTRICT_CONN_TAG_QSG.

For further information on WebSphere MQ connection options, see Connecting to a queue manager using the MQCONNX call

CONNTAG

A tag that the queue manager associates with the resources updated by the application within a unit of work while the application is connected to the queue manager.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: CONNTAG

JMS administration tool short name: CNTAG

Programmatic access

Setters/getters

- MQConnectionFactory.setConnTag()
- MQConnectionFactory.getConnTag()

Values

A byte array of 128 elements, where each element is 0
This is the default value.

Any string
The value is truncated if it is longer than 128 bytes.

DESCRIPTION

A description of the stored object.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, Queue, Topic, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: DESCRIPTION

JMS administration tool short name: DESC

Programmatic access

Setters/getters

- MQConnectionFactory.setDescription()
- MQConnectionFactory.getDescription()

Values

null This is the default value.

Any valid string

DIRECTAUTH

Whether SSL authentication is used on a real-time connection to a broker.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: DIRECTAUTH

JMS administration tool short name: DAUTH

Programmatic access

Setters/getters

- MQConnectionFactory.setDirectAuth()
- MQConnectionFactory.getDirectAuth()

Values

BASIC
No authentication, username authentication, or password authentication. This is the default value.

CERTIFICATE
Public key certificate authentication.

ENCODING

How numeric data in the body of a message is represented when the message is sent to this destination. The property specifies the representation of binary integers, packed decimal integers, and floating point numbers.

Applicable Objects

Queue, Topic

JMS administration tool long name: ENCODING

JMS administration tool short name: ENC

Programmatic access

Setters/getters

- MQDestination.setEncoding()
- MQDestination.getEncoding()

Values

ENCODING property

The valid values that the `ENCODING` property can take are constructed from the three sub-properties:

integer encoding
Either normal or reversed

decimal encoding
Either normal or reversed

floating-point encoding
IEEE normal, IEEE reversed, or z/OS

The `ENCODING` property is expressed as a three-character string with the following syntax:

```
{N|R}{N|R}{N|R|3}
```

In this string:

- `N` denotes normal
- `R` denotes reversed
- `3` denotes z/OS
- The first character represents *integer encoding*
- The second character represents *decimal encoding*
- The third character represents *floating-point encoding*

This provides a set of twelve possible values for the `ENCODING` property.

There is an additional value, the string `NATIVE`, which sets appropriate encoding values for the Java platform.

The following examples show valid combinations for `ENCODING`:

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

EXPIRY

The time after which messages at a destination expire.

Applicable Objects

Queue, Topic

JMS administration tool long name: EXPIRY

JMS administration tool short name: EXP

Programmatic access

Setters/getters

- MQDestination.setExpiry()
- MQDestination.getExpiry()

Values

APP Expiry can be defined by the JMS application. This is the default value.

UNLIM
No expiry occurs.

0 No expiry occurs.

Any positive integer representing expiry in milliseconds.

FAILIFQUIESCE

This property determines whether calls to certain methods fail if the queue manager is in a quiescing state.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, Queue, Topic, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: FAILIFQUIESCE

JMS administration tool short name: FIQ

Programmatic access

Setters/getters

- MQConnectionFactory.setFailIfQuiesce()
- MQConnectionFactory.getFailIfQuiesce()

Values

YES Calls to certain methods fail if the queue manager is in a quiescing state. If an application detects that the queue manager is quiescing, the application can complete its immediate task and close the connection, allowing the queue manager to stop. This is the default value.

NO No method call fails because the queue manager is in a quiescing state. If you specify this value, an application cannot detect that the queue manager is quiescing. The application might continue to perform operations against the queue manager, and therefore prevent the queue manager from stopping.

HOSTNAME

For a connection to a queue manager, the host name or IP address of the system on which the queue manager is running or, for a real-time connection to a broker, the host name or IP address of the system on which the broker is running.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: HOSTNAME

JMS administration tool short name: HOST

Programmatic access

Setters/getters

- MQConnectionFactory.setHostName()
- MQConnectionFactory.getHostName()

Values

localhost
This is the default value.

Any valid string

LOCALADDRESS

For a connection to a queue manager, this property specifies either the local network interface to be used, or the local port, or range of local ports, to be used. For a real-time connection to a broker, this property is relevant only when multicast is used, and specifies the local network interface to be used.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: LOCALADDRESS

JMS administration tool short name: LA

Programmatic access

Setters/getters

- MQConnectionFactory.setLocalAddress()
- MQConnectionFactory.getLocalAddress()

Values

"" (empty string)
This is the default value.

A string in the format [ip-addr][(low-port[,high-port])]
Here are some examples:

192.0.2.0

The channel binds to address 192.0.2.0 locally.

192.0.2.0(1000)

The channel binds to address 192.0.2.0 locally and uses port 1000.

192.0.2.0(1000,2000)

The channel binds to address 192.0.2.0 locally and uses a port in the range 1000 to 2000.

(1000)

The channel binds to port 1000 locally.

(1000,2000)

The channel binds to a port in the range 1000 to 2000 locally.

You can specify a host name instead of an IP address. For a real-time connection to a broker, this property is relevant only when multicast is used, and the value of the property must not contain a port number, or a range of port numbers. The only valid values of the property in this case are null, an IP address, or a host name.

MAPNAMESTYLE

Allows compatibility style to be used for MapMessage element names.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: MAPNAMESTYLE

JMS administration tool short name: MNST

Programmatic access

Setters/getters

- MQConnectionFactory.setMapNameStyle()
- MQConnectionFactory.getMapNameStyle()

Values

STANDARD

The standard com.ibm.jms.JMSMapMessage element naming format is to be used. This is the default value and allows non-legal Java identifiers to be used as the element name.

COMPATIBLE

The older com.ibm.jms.JMSMapMessage element naming format is to be used. Only legal Java identifiers can be used as the element name. This is needed only if map messages are being sent to an application that is using a version of IBM WebSphere MQ classes for JMS earlier than Version 5.3.

MAXBUFFSIZE

The maximum number of received messages that can be stored in an internal message buffer while waiting to be processed by the application. This property applies only when TRANSPORT has the value DIRECT or DIRECTHTTP.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: MAXBUFFSIZE

JMS administration tool short name: MBSZ

Programmatic access

Setters/getters

- MQConnectionFactory.setMaxBufferSize()
- MQConnectionFactory.getMaxBufferSize()

Values

1000 This is the default value.

Any positive integer

MDREAD

This property determines whether a JMS application can extract the values of MQMD fields.

Applicable Objects

JMS administration tool long name: MDREAD

JMS administration tool short name: MDR

Programmatic access

Setters/getters

- MQDestination.setMQMDReadEnabled()
- MQDestination.getMQMDReadEnabled()

Values

NO When sending messages, the JMS_IBM_MQMD* properties on a sent message are not updated to reflect the updated field values in the MQMD. When receiving messages, none of the JMS_IBM_MQMD* properties are available on a received message, even if the sender had set some or all of them. This is the default value for administrative tools.

For programs, use False.

Yes When sending messages, all of the JMS_IBM_MQMD* properties on a sent message are updated to reflect the updated field values in the MQMD, including the properties that the sender did not set explicitly. When receiving messages, all of the JMS_IBM_MQMD* properties are available on a received message, including the properties that the sender did not set explicitly.

For programs, use True.

MDWRITE

This property determines whether a JMS application can set the values of MQMD fields.

Applicable Objects

Queue, Topic

JMS administration tool long name: MDWRITE

JMS administration tool short name: MDR

Programmatic access

Setters/getters

- MQDestination.setMQMDWriteEnabled()
- MQDestination.getMQMDWriteEnabled()

Values

NO All JMS_IBM_MQMD* properties are ignored and their values are not copied into the underlying MQMD structure. This is the default value for administrative tools.

For programs, use False.

YES JMS_IBM_MQMD* properties are processed. Their values are copied into the underlying MQMD structure.

For programs, use True.

MDMSGCTX

What level of message context is to be set by the JMS application. The application must be running with appropriate context authority for this property to take effect.

Applicable Objects

JMS administration tool long name: MDMSGCTX

JMS administration tool short name: MDCTX

Programmatic access

Setters/getters

- MQDestination.setMQMDMessageContext()
- MQDestination.getMQMDMessageContext()

Values

DEFAULT

The MQOPEN API call and the MQPMO structure specify no explicit message context options. This is the default value for administrative tools.

For programs, use WMQ_MDCTX_DEFAULT.

SET_IDENTITY_CONTEXT

The MQOPEN API call specifies the message context option MQOO_SET_IDENTITY_CONTEXT and the MQPMO structure specifies MQPMO_SET_IDENTITY_CONTEXT.

For programs, use WMQ_MDCTX_SET_IDENTITY_CONTEXT.

SET_ALL_CONTEXT
The MQOPEN API call specifies the message context option MQOO_SET_ALL_CONTEXT and the MQPMO structure specifies MQPMO_SET_ALL_CONTEXT.

For programs, use WMQ_MDCTX_SET_ALL_CONTEXT.

MSGBATCHSZ

The maximum number of messages to be taken from a queue in one packet when using asynchronous message delivery.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: MAXBUFFSIZE

JMS administration tool short name: MBSZ

Programmatic access

Setters/getters

- MQConnectionFactory.setMsgBatchSize()
- MQConnectionFactory.getMsgBatchSize()

Values

10 This is the default value.

Any positive integer

MSGBODY

Determines whether a JMS application accesses the MQRFH2 of a IBM WebSphere MQ message as part of the message payload.

Applicable Objects

Queue, Topic

JMS administration tool long name: WMQ_MESSAGE_BODY

JMS administration tool short name: MBODY

Programmatic access

Setters/getters

- MQConnectionFactory.setMessageBodyStyle()
- MQConnectionFactory.getMessageBodyStyle()

Values

UNSPECIFIED
When sending, IBM WebSphere MQ classes for JMS does or does not generate and include an MQRFH2 header, depending on the value of WMQ_TARGET_CLIENT. When receiving, acts as value `JMS`.

JMS When sending, IBM WebSphere MQ classes for JMS automatically generates an MQRFH2 header and includes it in the WebSphere MQ message.

When receiving, IBM WebSphere MQ classes for JMS set the JMS message properties according to values in the MQRFH2 (if present); it does not present the MQRFH2 as part of the JMS message body.

MQ When sending, IBM WebSphere MQ classes for JMS does not generate an MQRFH2.

When receiving, IBM WebSphere MQ classes for JMS presents the MQRFH2 as part of the JMS message body.

MSGRETENTION

Whether the connection consumer keeps undelivered messages on the input queue.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory,

JMS administration tool long name: MSGRETENTION

JMS administration tool short name: MRET

Programmatic access

Setters/getters

- MQConnectionFactory.setMessageRetention()
- MQConnectionFactory.getMessageRetention()

Values

Yes Undelivered messages remain on the input queue. This is the default value.

No Undelivered messages are dealt with according to their disposition options.

MSGSELECTION

Determines whether message selection is done by the WebSphere MQ classes for JMS or by the broker. If TRANSPORT has the value DIRECT, message selection is always done by the broker and the value of MSGSELECTION is ignored. Message selection by the broker is not supported when BROKERVER has the value V1.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: MSGSELECTION

JMS administration tool short name: MSEL

Programmatic access

Setters/getters

- MQConnectionFactory.setMessageSelection()
- MQConnectionFactory.getMessageSelection()

Values

CLIENT
Message selection is done by WebSphere MQ classes for JMS. This is the default value.

BROKER
Message selection is done by the broker.

MULTICAST

To enable multicast on a real-time connection to a broker and, if enabled, to specify the precise way in which multicast is used to deliver messages from the broker to a message consumer. The property has no effect on how a message producer sends messages to a broker.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, Topic

JMS administration tool long name: MULTICAST

JMS administration tool short name: MCAST

Programmatic access

Setters/getters

- MQConnectionFactory.setMulticast()
- MQConnectionFactory.getMulticast()

Values

DISABLED
Messages are not delivered to a message consumer using multicast transport. This is the default value for ConnectionFactory and TopicConnectionFactory objects.

ASCF Messages are delivered to a message consumer according to the multicast setting for the connection factory associated with the message consumer. The multicast setting for the connection factory is noted at the time that the message consumer is created. This value is valid only for Topic objects, and is the default value for Topic objects.

ENABLED
If the topic is configured for multicast in the broker, messages are delivered to a message consumer using multicast transport. A reliable quality of service is used if the topic is configured for reliable multicast.

RELIABLE
If the topic is configured for reliable multicast in the broker, messages are delivered to the message consumer using multicast transport with a reliable quality of service. If the topic is not configured for reliable multicast, you cannot create a message consumer for the topic.

NOTR
If the topic is configured for multicast in the broker, messages are delivered to the message consumer using multicast transport. A reliable quality of service is not used even if the topic is configured for reliable multicast.

OPTIMISTICPUBLICATION

This property determines whether WebSphere MQ classes for JMS returns control immediately to a publisher that has published a message, or whether it returns control only after it has completed all the processing associated with the call and can report the outcome to the publisher.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: OPTIMISTICPUBLICATION

JMS administration tool short name: OPTPUB

Programmatic access

Setters/getters

- MQConnectionFactory.setOptimisticPublication()
- MQConnectionFactory.getOptimisticPublication()

Values

NO When a publisher publishes a message, WebSphere MQ classes for JMS do not return control to the publisher until it has completed all the processing associated with the call and can report the outcome to the publisher. This is the default value.

YES When a publisher publishes a message, WebSphere MQ classes for JMS returns control to the publisher immediately, before it has completed all the processing associated with the call and can report the outcome to the publisher. WebSphere MQ classes for JMS reports the outcome only when the publisher commits the message.

OUTCOMENOTIFICATION

This property determines whether WebSphere MQ classes for JMS return control immediately to a subscriber that has just acknowledged or committed a message, or whether it returns control only after it has completed all the processing associated with the call and can report the outcome to the subscriber.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: OUTCOMENOTIFICATION

JMS administration tool short name: NOTIFY

Programmatic access

Setters/getters

- MQConnectionFactory.setOutcomeNotification()
- MQConnectionFactory.getOutcomeNotification()

Values

YES When a subscriber acknowledges or commits a message, WebSphere MQ classes for JMS do not return control to the subscriber until it has completed all the processing associated with the call and can report the outcome to the subscriber. This is the default value.

NO When a subscriber acknowledges or commits a message, WebSphere MQ classes for JMS returns

control to the subscriber immediately, before it has completed all the processing associated with the call and can report the outcome to the subscriber.

PERSISTENCE

The persistence of messages sent to a destination.

Applicable Objects

Queue, Topic

JMS administration tool long name: PERSISTENCE

JMS administration tool short name: PER

Programmatic access

Setters/getters

- MQDestination.setPersistence()
- MQDestination.getPersistence()

Values

APP Persistence is defined by the JMS application. This is the default value.

QDEF Persistence takes the value of the queue default.

PERS Messages are persistent.

NON Messages are nonpersistent.

HIGH See JMS persistent messages for further information on the use of this value.

POLLINGINT

If each message listener within a session has no suitable message on its queue, this is the maximum interval, in milliseconds, that elapses before each message listener tries again to get a message from its queue. If it frequently happens that no suitable message is available for any of the message listeners in a session, consider increasing the value of this property. This property is relevant only if TRANSPORT has the value BIND or CLIENT.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: POLLINGINT

JMS administration tool short name: PINT

Programmatic access

Setters/getters

- MQConnectionFactory.setPollingInterval()
- MQConnectionFactory.getPollingInterval()

Values

5000 This is the default value.

Any positive integer

PORT

For a connection to a queue manager, the number of the port on which the queue manager is listening or, for a real-time connection to a broker, the number of the port on which the broker is listening for real-time connections.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: PORT

JMS administration tool short name: PORT

Programmatic access

Setters/getters

- MQConnectionFactory.setPort()
- MQConnectionFactory.getPort()

Values

1414 This is the default value if TRANSPORT is set to CLIENT.

1506 This is the default value if TRANSPORT is set to DIRECT or DIRECTHTTP.

Any positive integer

PRIORITY

The priority for messages sent to a destination.

Applicable Objects

Queue, Topic

JMS administration tool long name: PRIORITY

JMS administration tool short name: PRI

Programmatic access

Setters/getters

- MQDestination.setPriority()
- MQDestination.getPriority()

Values

APP Priority is defined by the JMS application. This is the default value.

QDEF Priority takes the value of the queue default.

Any integer in the range 0-9
Lowest to highest.

PROCESSDURATION

This property determines whether a subscriber guarantees to process quickly any message it receives before returning control to WebSphere MQ classes for JMS.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: PROCESSDURATION

JMS administration tool short name: PROCDUR

Programmatic access

Setters/getters

- MQConnectionFactory.setProcessDuration()
- MQConnectionFactory.getProcessDuration()

Values

UNKNOWN
A subscriber can give no guarantee about how quickly it can process any message it receives. This is the default value.

SHORT
A subscriber guarantees to process quickly any message it receives before returning control to WebSphere MQ classes for JMS.

PROVIDERVERSION

This property differentiates between the two WebSphere MQ messaging modes of operation: WebSphere MQ messaging provider normal mode and WebSphere MQ messaging provider migration mode.

The WebSphere MQ messaging provider normal mode uses all the features of the WebSphere MQ Version 7.0 queue managers to implement JMS. This mode is used only to connect to a WebSphere MQ queue manager and can connect to WebSphere MQ Version 7.0 queue managers in either client or bindings mode. This mode is optimized to use the new WebSphere MQ Version 7.0 function. If you are not using WebSphere MQ Real-Time Transport, then the mode of operation used is determined primarily by the PROVIDERVERSION property of the connection factory.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnection Factory, XATopicConnectionFactory

JMS administration tool long name: PROVIDERVERSION

JMS administration tool short name: PVER

Programmatic access

Setters/getters

- MQConnectionFactory.setProviderVersion()
- MQConnectionFactory.getProviderVersion()

Values

You can set **PROVIDERVERSION** to the possible values: `7`, `6`, or *unspecified*. However, **PROVIDERVERSION** can be a string in any one of the following formats:

- V.R.M.F
- V.R.M
- V.R
- V

where V, R, M and F are integer values greater than or equal to zero.

7 Uses the WebSphere MQ messaging provider normal mode.

If you set PROVIDERVERSION to 7 only the WebSphere MQ messaging provider normal mode of operation is available. If the queue manager that is connected to as a result of the other settings in the connection factory is not a Version 7.0 queue manager, the createConnection() method fails with an exception.

The WebSphere MQ messaging provider normal mode uses the sharing conversations feature and the number of conversations that can be shared is controlled by the SHARECNV() property on the server connection channel. If this property is set to 0, you cannot use WebSphere MQ messaging provider normal mode and the createConnection() method fails with an exception.

6 Uses the WebSphere MQ messaging provider migration mode.

The WebSphere MQ classes for JMS use the features and algorithms supplied with WebSphere MQ Version 6.0. If you want to connect to WebSphere Event Broker or WebSphere Message Broker using WebSphere MQ Enterprise Transport, you must use this mode. You can connect to a WebSphere MQ Version 7.0 queue manager using this mode, but none of the new features of a Version 7.0 queue manager are used, for example, read ahead or streaming.

unspecified
This is the default value and the actual text is "unspecified".

A connection factory that was created with a previous version of WebSphere MQ classes for JMS in JNDI takes this value when the connection factory is used with the new version of WebSphere MQ classes for JMS. The following algorithm is used to determine which mode of operation is used. This algorithm is used when the createConnection() method is called and uses other aspects of the connection factory to determine if WebSphere MQ messaging provider normal mode or WebSphere MQ messaging provider migration mode is required.

- First, an attempt to use WebSphere MQ messaging provider normal mode is made.
- If the queue manager connected is not WebSphere MQ Version 7.0, the connection is closed and WebSphere MQ messaging provider migration mode is used instead.
- If the SHARECNV() property on the server connection channel is set to 0, the connection is closed and WebSphere MQ messaging provider migration mode is used instead.
- If BROKERVER is set to 1 or the new default "unspecified" value, WebSphere MQ messaging provider normal mode continues to be used, and therefore any publish/subscribe operations use the new WebSphere MQ V7.0 features. If WebSphere Event Broker or WebSphere Message Broker are used in compatibility mode (and you want to use Version 6.0 publish/subscribe function rather than the WebSphere MQ Version 7 publish/subscribe function), set PROVIDERVERSION to 6 ensure WebSphere MQ messaging provider migration mode is used.

PROXYHOSTNAME

The host name or IP address of the system on which the proxy server is running when using a real-time connection to a broker through a proxy server.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: PROXYHOSTNAME

JMS administration tool short name: PHOST

Programmatic access

Setters/getters

- MQConnectionFactory.setProxyHostName()
- MQConnectionFactory.getProxyHostName()

Values

null The host name of the proxy server. This is the default value.

PROXYPORT

The number of the port on which the proxy server is listening when using a real-time connection to a broker through a proxy server.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: PROXYPORT

JMS administration tool short name: PPORT

Programmatic access

Setters/getters

MQConnectionFactory.setProxyPort()

MQConnectionFactory.getProxyPort()

Values

443 The port number of the proxy server. This is the default value.

PUBACKINT

The number of messages published by a publisher before WebSphere MQ classes for JMS requests an acknowledgment from the broker.

When you lower the value of this property, WebSphere MQ classes for JMS requests acknowledgments more often, therefore the performance of the publisher decreases. When you raise the value, WebSphere MQ classes for JMS take a longer time to throw an exception if the broker fails. This property is relevant only if TRANSPORT has the value BIND or CLIENT.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: PROXYPORT

JMS administration tool short name: PPORT

Programmatic access

Setters/getters

MQConnectionFactory.setPubAckInterval()

MQConnectionFactory.getPubAckInterval()

Values

25 Any positive integer may be the default value.

PUTASYNCALLOWED

This property determines whether message producers are allowed to use asynchronous puts to send messages to this destination.

Applicable Objects

Queue, Topic

JMS administration tool long name: PUTASYNCALLOWED

JMS administration tool short name: PAALD

Programmatic access

Setters/getters

MQDestination.setPutAsyncAllowed()

MQDestination.getPutAsyncAllowed()

Values

AS_DEST
Determine whether asynchronous puts are allowed by referring to the queue or topic definition. This is the default value.

AS_Q_DEF
Determine whether asynchronous puts are allowed by referring to the queue definition.

AS_TOPIC_DEF
Determine whether asynchronous puts are allowed by referring to the topic definition.

NO Asynchronous puts are not allowed.

YES Asynchronous puts are allowed.

QMANAGER

The name of the queue manager to connect to.

However, if your application uses a client channel definition table to connect to a queue manager, see Using a client channel definition table with WebSphere MQ classes for JMS.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, Queue, XAConnectionFactory, XAQueueConnection Factory, XATopicConnectionFactory

JMS administration tool long name: QMANAGER

JMS administration tool short name: QMGR

Programmatic access

Setters/getters

- MQConnectionFactory.setQueueManager()
- MQConnectionFactory.getQueueManager()

Values

" " (empty string)
Any string can be the default value.

QUEUE

The name of the JMS queue destination. This matches the name of the queue used by the queue manager.

Applicable Objects

Queue

JMS administration tool long name: QUEUE

JMS administration tool short name: QU

Values

Any string
Any valid IBM WebSphere MQ queue name.

Related reference:

../com.ibm.mq.pro.doc/q003340_.dita
IBM WebSphere MQ object names have maximum lengths and are case-sensitive. Not all characters are supported for every object type, and many objects have rules concerning the uniqueness of names.

READAHEADALLOWED

This property determines whether message consumers and queue browsers are allowed to use read ahead to get nonpersistent messages from this destination into an internal buffer before receiving them.

Applicable Objects

Queue, Topic

JMS administration tool long name: READAHEADALLOWED

JMS administration tool short name: RAALD

Programmatic access

Setters/getters

- MQDestination.setReadAheadAllowed()
- MQDestination.getReadAheadAllowed()

Values

AS_DEST
Determine whether read ahead is allowed by referring to the queue or topic definition. This is the default value in administrative tools.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_AS_DEST in programs.

AS_Q_DEF
Determine whether read ahead is allowed by referring to the queue definition.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_AS_Q_DEF in programs.

AS_TOPIC_DEF
Determine whether read ahead is allowed by referring to the topic definition.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_AS_TOPIC_DEF in programs.

NO Read ahead is not allowed.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_DISABLED in programs.

YES Read ahead is allowed.

Use WMQConstants.WMQ_READ_AHEAD_ALLOWED_ENABLED in programs.

READAHEADCLOSEPOLICY

For messages being delivered to an asynchronous message listener, what happens to messages in the internal read ahead buffer when the message consumer is closed.

Applicable Objects

Queue, Topic

JMS administration tool long name: READAHEADCLOSEPOLICY

JMS administration tool short name: RACP

Programmatic access

Setters/getters

- MQDestination.setReadAheadClosePolicy()
- MQDestination.getReadAheadClosePolicy()

Values

DELIVER_ALL
All messages in the internal read ahead buffer are delivered to the message listener of the application before returning. This is the default value in administrative tools.

Use WMQConstants.WMQ_READ_AHEAD_DELIVERALL in programs.

DELIVER_CURRENT
Only the current message listener invocation completes before returning, potentially leaving messages in the internal read ahead buffer, which are then discarded.

Use WMQConstants.WMQ_READ_AHEAD_DELIVERCURRENT in programs.

RECEIVECCSID

Destination property that sets the target CCSID for queue manager message conversion. The value is ignored unless RECEIVECONVERSION is set to `WMQ_RECEIVE_CONVERSION_QMGR`

Applicable Objects

Queue, Topic

JMS administration tool long name: RECEIVECCSID

JMS administration tool short name: RCCS

Programmatic access

Setters/Getters

- MQDestination.setReceiveCCSID
- MQDestination.getReceiveCCSID

Values

`WMQConstants.WMQ_RECEIVE_CCSID_JVM_DEFAULT`
0 - Use JVM Charset.defaultCharset

1208 `UTF-8`

CCSID
Supported coded character set identifier.

RECEIVECONVERSION

Destination property that determines if data conversion is going to be performed by the queue manager.

Applicable Objects

Queue, Topic

JMS administration tool long name: RECEIVECONVERSION

JMS administration tool short name: RCNV

Programmatic access

Setters/Getters

- MQDestination.setReceiveConversion
- MQDestination.getReceiveConversion

Values

WMQConstants.WMQ_RECEIVE_CONVERSION_CLIENT_MSG
1 - Only perform data conversion on the JMS client. The default value from up to V7.0, and from, and including, 7.0.1.5.

WMQConstants.WMQ_RECEIVE_CONVERSION_QMGR
2 - Perform data conversion on the queue manager before sending a message to the client. The default (and only) value from V7.0 to V7.0.1.4 inclusive, except if APAR IC72897 is applied.

RECEIVEISOLATION

This property determines whether a subscriber might receive messages that have not been committed on the subscriber queue.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: RECEIVEISOLATION

JMS administration tool short name: RCVISOL

Values

COMMITTED
A subscriber receives only those messages on the subscriber queue that have been committed. This is the default value in administrative tools.

Use WMQConstants.WMQ_RCVISOL_COMMITTED in programs.

UNCOMMITTED
A subscriber can receive messages that have not been committed on the subscriber queue.

Use WMQConstants.WMQ_RCVISOL_UNCOMMITTED in programs.

RECEXIT

Identifies a channel receive exit, or a sequence of receive exits to be run in succession.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: RECEXIT

JMS administration tool short name: RCX

Programmatic access

Setters/getters

- MQConnectionFactory.setReceiveExit()
- MQConnectionFactory.getReceiveExit()

Values

null A string comprising one or more items separated by commas, where each item is either:

- The name of a class that implements the WMQReceiveExit interface (for a channel receive exit written in Java).
- A string in the format *libraryName(entryPointName)* (for a channel receive exit not written in Java).

This is the default value.

RECEXITINIT

The user data that is passed to channel receive exits when they are called.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: RECEXITINIT

JMS administration tool short name: RCXI

Programmatic access

Setters/getters

- MQConnectionFactory.setReceiveExitInit()
- MQConnectionFactory.getReceiveExitInit()

Values

null A string comprising one or more items of user data separated by commas. This is the default value.

REPLYTOSTYLE

Determines how the JMSReplyTo field in a received message is constructed.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: REPLYTOSTYLE

JMS administration tool short name: RTOST

Programmatic access

Setters/getters

- MQConnectionFactory.setReplyToStyle()
- MQConnectionFactory.getReplyToStyle()

Values

DEFAULT
Equivalent to `MQMD`.

RFH2 Use the value supplied in the RFH2 header. If a JMSReplyTo value has been set in the sending application, use that value.

MQMD
Use the MQMD supplied value. This behavior is equivalent to the default behavior of WebSphere MQ Version 6.0.2.4 and 6.0.2.5.

If the JMSReplyTo value set by the sending application does not contain a queue manager name, the receiving queue manager inserts its own name in the MQMD. If you set this parameter to `MQMD`, the reply-to queue you use is on the receiving queue manager. If you set this parameter to `RFH2`, the reply-to queue you use is on the queue manager specified in the RFH2 of the sent message as originally set by the sending application.

If the JMSReplyTo value set by the sending application contains a queue manager name, the value of this parameter is unimportant because both the MQMD and RFH2 contain the same value.

RESCANINT

When a message consumer in the point-to-point domain uses a message selector to select which messages it wants to receive, WebSphere MQ classes for JMS search the WebSphere MQ queue for suitable messages in the sequence determined by the `MsgDeliverySequence` attribute of the queue.

After WebSphere MQ classes for JMS find a suitable message and deliver it to the consumer, WebSphere MQ classes for JMS resume the search for the next suitable message from its current position in the queue. WebSphere MQ classes for JMS continue to search the queue in this way until it reaches the end of the queue, or until the interval of time in milliseconds, as determined by the value of this property, has expired. In each case, WebSphere MQ classes for JMS return to the beginning of the queue to continue the search, and a new time interval commences.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory

JMS administration tool long name: RESCANINT

JMS administration tool short name: RINT

Programmatic access

Setters/getters

- MQConnectionFactory.setRescanInterval()
- MQConnectionFactory.getRescanInterval()

Values

5000 Any positive integer can be the default value.

SECEXIT

Identifies a channel security exit.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SECEXIT

JMS administration tool short name: SXC

Programmatic access

Setters/getters

- MQConnectionFactory.setSecurityExit()
- MQConnectionFactory.getSecurityExit()

Values

null The name of a class that implements the WMQSecurityExit interface (for a channel security exit written in Java).

A string in the format *libraryName(entryPointName)* (for a channel security exit not written in Java).

SECEXITINIT

The user data that is passed to a channel security exit when it is called.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SECEXITINIT

JMS administration tool short name: SCXI

Programmatic access

Setters/getters

- MQConnectionFactory.setSecurityExitInit()
- MQConnectionFactory.getSecurityExitInit()

Values

null Any string can be the default value.

SENDCHECKCOUNT

The number of send calls to allow between checking for asynchronous put errors, within a single non-transacted JMS session.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SENDCHECKCOUNT

JMS administration tool short name: SCC

Programmatic access

Setters/getters

- MQConnectionFactory.setSendCheckCount()
- MQConnectionFactory.getSendCheckCount()

Values

null Any string can be the default value.

SENDEXIT

Identifies a channel send exit, or a sequence of send exits to be run in succession.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SENDEXIT

JMS administration tool short name: SDX

Programmatic access

Setters/getters

- MQConnectionFactory.setSendExit()
- MQConnectionFactory.getSendExit()

Values

null Any string comprising one or more items separated by commas, where each item is either:

- The name of a class that implements the WMQSendExit interface (for a channel send exit written in Java).
- A string in the format *libraryName(entryPointName)* (for a channel send exit not written in Java).
-

This is the default value.

SENDEXITINIT

The user data that is passed to channel send exits when they are called.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SENDEXITINIT

JMS administration tool short name: SDXI

Programmatic access

Setters/getters

- MQConnectionFactory.setSendExitInit()
- MQConnectionFactory.getSendExitInit()

Values

null Any string comprising one or more items of user data separated by commas can be the default value.

SHARECONVALLOWED

This property determines whether a client connection can share its socket with other top-level JMS connections from the same process to the same queue manager, if the channel definitions match.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SHARECONVALLOWED

JMS administration tool short name: SCALD

Programmatic access

Setters/getters

- MQConnectionFactory.setShareConvAllowed()
- MQConnectionFactory.getShareConvAllowed()

Values

YES This is the default value for administrative tools.

For programs, use WMQConstants.WMQ_SHARE_CONV_ALLOWED_YES.

NO This value is for administrative tools.

For programs, use WMQConstants.WMQ_SHARE_CONV_ALLOWED_NO.

SPARSESUBS

Controls the message retrieval policy of a TopicSubscriber object.

Applicable Objects

ConnectionFactory, TopicConnectionFactory

JMS administration tool long name: SPARSESUBS

JMS administration tool short name: SSUBS

Programmatic access

Setters/getters

- MQConnectionFactory.setSparseSubscriptions()
- MQConnectionFactory.getSparseSubscriptions()

Values

NO Subscriptions receive frequent matching messages. This is the default value for administrative tools.

For programs, use false.

YES Subscriptions receive infrequent matching messages. This value requires that the subscription queue can be opened for browse.

For programs, use true.

SSLCIPHERSUITE

The CipherSuite to use for an SSL connection.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLCIPHERSUITE

JMS administration tool short name: SCPHS

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLCipherSuite()
- MQConnectionFactory.getSSLCipherSuite()

Values

null This is the default value. For more information, see SSL properties of JMS objects.

SSLCRL

CRL servers to check for SSL certificate revocation.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLCRL

JMS administration tool short name: SCRL

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLCertStores()
- MQConnectionFactory.getSSLCertStores()

Values

null Space-separated list of LDAP URLs. This is the default value. For more information, see SSL properties of JMS objects.

SSLFIPSREQUIRED

This property determines whether an SSL connection must use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS).

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLFIPSREQUIRED

JMS administration tool short name: SFIPS

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLFipsRequired()
- MQConnectionFactory.getSSLFipsRequired()

Values

NO An SSL connection can use any CipherSuite that is not supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS).

This is the default value. In programs, use false.

YES An SSL connection must use a CipherSuite that is supported by IBMJSSEFIPS.

In programs, use true.

SSLPEERNAME

For SSL, a *distinguished name* skeleton that must match that provided by the queue manager.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLPEERNAME

JMS administration tool short name: SPEER

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLPeerName()
- MQConnectionFactory.getSSLPeerName()

Values

null This is the default value. For more information, see SSL properties of JMS objects.

SSLRESETCOUNT

For SSL, the total number of bytes sent and received by a connection before the secret key that is used for encryption is renegotiated.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SSLRESETCOUNT

JMS administration tool short name: SRC

Programmatic access

Setters/getters

- MQConnectionFactory.setSSLResetCount()
- MQConnectionFactory.getSSLResetCount()

Values

0 Zero, or any positive integer less than or equal to 999, 999, 999. This is the default value. For more information, see SSL properties of JMS objects.

STATREFRESHINT

The interval, in milliseconds, between refreshes of the long running transaction that detects when a subscriber loses its connection to the queue manager.

This property is relevant only if SUBSTORE has the value QUEUE.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: STATREFRESHINT

JMS administration tool short name: SRI

Programmatic access

Setters/getters

- MQConnectionFactory.setStatusRefreshInterval()
- MQConnectionFactory.getStatusRefreshInterval()

Values

6000 Any positive integer can be the default value. For more information, see SSL properties of JMS objects.

SUBSTORE

Where WebSphere MQ classes for JMS stores persistent data relating to active subscriptions.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SUBSTORE

JMS administration tool short name: SS

Programmatic access

Setters/getters

- MQConnectionFactory.setSubscriptionStore()
- MQConnectionFactory.getSubscriptionStore()

Values

BROKER
Use the broker-based subscription store to hold details of subscriptions. This is the default value for administrative tools.

For programs, use WMQConstants.WMQ_SUBSTORE_BROKER.

MIGRATE
Transfer subscription information from the queue-based subscription store to the broker-based subscription store.

For programs, use WMQConstants.WMQ_SUBSTORE_MIGRATE.

QUEUE
Use the queue-based subscription store to hold details of subscriptions.

For programs, use WMQConstants.WMQ_SUBSTORE_QUEUE.

SYNCPOINTALLGETS

This property determines whether all gets are to be performed under syncpoint.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: SYNCPOINTALLGETS

JMS administration tool short name: SPAG

Programmatic access

Setters/getters

- MQConnectionFactory.setSyncpointAllGets()
- MQConnectionFactory.getSyncpointAllGets()

Values

No This is the default value.

Yes

TARGCLIENT

This property determines whether the WebSphere MQ RFH2 format is used to exchange information with target applications.

Applicable Objects

Queue, Topic

JMS administration tool long name: TARGCLIENT

JMS administration tool short name: TC

Programmatic access

Setters/getters

- MQDestination.setTargetClient()
- MQDestination.getTargetClient()

Values

JMS The target of the message is a JMS application. This is the default value for administrative tools.

For programs, use WMQConstants.WMQ_CLIENT_JMS_COMPLIANT.

MQ The target of the message is a non-JMS WebSphere MQ application.

For programs, use WMQConstants.WMQ_CLIENT_NONJMS_MQ.

TARGCLIENTMATCHING

This property determines whether a reply message, sent to the queue identified by the JMSReplyTo header field of an incoming message, has an MQRFH2 header only if the incoming message has an MQRFH2 header.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory

JMS administration tool long name: TARGCLIENTMATCHING

JMS administration tool short name: TCM

Programmatic access

Setters/getters

- MQConnectionFactory.setTargetClientMatching()
- MQConnectionFactory.getTargetClientMatching()

Values

YES If an incoming message does not have an MQRFH2 header, the TARGCLIENT property of the Queue object derived from the JMSReplyTo header field of the message is sent to MQ. If the message does have an MQRFH2 header, the TARGCLIENT property is set to JMS instead. This is the default value for administrative tools.

For programs, use true.

NO The TARGCLIENT property of the Queue object derived from the JMSReplyTo header field of an incoming message is always set to JMS.

For programs, use false.

TEMPMODEL

The name of the model queue from which JMS temporary queues are created.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory

JMS administration tool long name: TEMPMODEL

JMS administration tool short name: TM

Programmatic access

Setters/getters

- MQConnectionFactory.setTemporaryModel()
- MQConnectionFactory.getTemporaryModel()

Values

SYSTEM.DEFAULT.MODEL.QUEUE

Any string can be the default value.

TEMPQPREFIX

The prefix that is used to form the name of a WebSphere MQ dynamic queue.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory

JMS administration tool long name: TEMPQPREFIX

JMS administration tool short name: TQP

Programmatic access

Setters/getters

- MQConnectionFactory.setTempQPrefix()
- MQConnectionFactory.getTempQPrefix()

Values

" " (empty string)
The prefix used is `CSQ.*` on z/OS and `AMQ.*` on all other platforms. These are the default values.

queue prefix
The queue prefix is any string that conforms to the rules for forming contents of the *DynamicQName* field in a WebSphere MQ object descriptor (structure MQOD), but the last non-blank character must be an asterisk.

TEMPTOPICPREFIX

When creating temporary topics, JMS generates a topic string of the form "TEMP/*TEMPTOPICPREFIX*/*unique_id*", or if this property is left with the default value, just "TEMP/*unique_id*". Specifying a non-empty TEMPTOPICPREFIX allows specific model queues to be defined for creating the managed queues for subscribers to temporary topics created under this connection.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: TEMPTOPICPREFIX

JMS administration tool short name: TTP

Programmatic access

Setters/getters

- MQConnectionFactory.setTempTopicPrefix()
- MQConnectionFactory.getTempTopicPrefix()

Values

Any non-null string consisting only of valid characters for a WebSphere MQ topic string. The default value is " " (empty string).

TOPIC

The name of the JMS topic destination, this value is used by the queue manager as the topic string of a publication or subscription.

Applicable Objects

Topic

JMS administration tool long name: TOPIC

JMS administration tool short name: TOP

Values

Any string

A string that forms a valid IBM WebSphere MQ topic string. When using IBM WebSphere MQ as a messaging provider with WebSphere Application Server, specify a value that matches the name by which the topic is known for administrative purposes within WebSphere Application Server.

Related information:

Topic strings

TRANSPORT

The nature of a connection to a queue manager or broker.

Applicable Objects

ConnectionFactory, QueueConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XAQueueConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: TRANSPORT

JMS administration tool short name: TRAN

Programmatic access

Setters/getters

- MQConnectionFactory.setTransportType()
- MQConnectionFactory.getTransportType()

Values

BIND For a connection to a queue manager in bindings mode. This is the default value for administrative tools.

For programs, use WMQConstants.WMQ_CM_BINDINGS.

CLIENT

For a connection to a queue manager in client mode.

For programs, use WMQConstants.WMQ_CM_CLIENT.

DIRECT

For a real-time connection to a broker not using HTTP tunnelling.

For programs, use WMQConstants.WMQ_CM_DIRECT_TCPIP.

DIRECTHTTP

For a real-time connection to a broker using HTTP tunnelling. Only HTTP 1.0 is supported.

For programs, use WMQConstants.WMQ_CM_DIRECT_HTTP.

WILDCARDFORMAT

This property determines which version of wildcard syntax is to be used.

Applicable Objects

ConnectionFactory, TopicConnectionFactory, XAConnectionFactory, XATopicConnectionFactory

JMS administration tool long name: WILDCARDFORMAT

JMS administration tool short name: WCFMT

Programmatic access

Setters/getters

- MQConnectionFactory.setWildCardFormat()
- MQConnectionFactory.getWildCardFormat()

Values

TOPIC_ONLY

Recognizes topic level wildcards only, as used in broker version 2. This is the default value for administrative tools.

For programs, use WMQConstants.WMQ_WILDCARD_TOPIC_ONLY.

CHAR_ONLY

Recognizes character wildcards only, as used in broker version 1.

For programs, use WMQConstants.WMQ_WILDCARD_CHAR_ONLY.

Dependencies between properties of WebSphere MQ classes for JMS objects

The validity of some properties is dependent on the particular values of other properties.

This dependency can occur in the following groups of properties:

- Client properties
- Properties for a real-time connection to a broker
- Exit initialization strings

Client properties

For a connection to a queue manager, the following properties are relevant only if TRANSPORT has the value CLIENT:

- HOSTNAME
- PORT
- CHANNEL
- LOCALADDRESS
- CCDTURL
- CCSID
- COMPHDR
- COMPMSG
- RECEXIT
- RECEXITINIT
- SECEXIT

- SECEXITINIT
- SENDEXIT
- SENDEXITINIT
- SHARECONVALLOWED
- SSLCIPHERSUITE
- SSLCRL
- SSLFIPSREQUIRED
- SSLPEERNAME
- SSLRESETCOUNT
- APPLICATIONNAME

You cannot set values for these properties by using the administration tool if TRANSPORT has the value BIND.

If TRANSPORT has the value CLIENT, the default value of the BROKERVER property is V1 and the default value of the PORT property is 1414. If you set the value of BROKERVER or PORT explicitly, a later change to the value of TRANSPORT does not override your choices.

Properties for a real-time connection to a broker

Only the following properties are relevant if TRANSPORT has the value DIRECT or DIRECTHTTP:

- BROKERVER
- CLIENTID
- DESCRIPTION
- DIRECTAUTH
- HOSTNAME
- LOCALADDRESS
- MAXBUFFSIZE
- MULTICAST (supported only for DIRECT)
- PORT
- PROXYHOSTNAME (supported only for DIRECT)
- PROXYPORT (supported only for DIRECT)

If TRANSPORT has the value DIRECT or DIRECTHTTP, the default value of the BROKERVER property is V2, and the default value of the PORT property is 1506. If you set the value of BROKERVER or PORT explicitly, a later change to the value of TRANSPORT does not override your choices.

Exit initialization strings

Do not set any of the exit initialization strings without supplying the corresponding exit name. The exit initialization properties are:

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

For example, specifying `RECEXITINIT(myString)` without specifying `RECEXIT(some.exit.classname)` causes an error.

The ENCODING property

The ENCODING property comprises three sub-properties, in twelve possible combinations.

The valid values that the ENCODING property can take are constructed from the three sub-properties:

integer encoding
Either normal or reversed

decimal encoding
Either normal or reversed

floating-point encoding
IEEE normal, IEEE reversed, or z/OS

The ENCODING property is expressed as a three-character string with the following syntax:
`{N|R}{N|R}{N|R|3}`

In this string:

- N denotes normal
- R denotes reversed
- 3 denotes z/OS
- The first character represents *integer encoding*
- The second character represents *decimal encoding*
- The third character represents *floating-point encoding*

This provides a set of twelve possible values for the ENCODING property.

There is an additional value, the string NATIVE, which sets appropriate encoding values for the Java platform.

The following examples show valid combinations for ENCODING:

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

SSL properties of JMS objects

Enable Secure Sockets Layer (SSL) encryption using the SSLCIPHERSUITE property. You can then change the characteristics of the SSL encryption using several other properties.

When you specify TRANSPORT(CLIENT), you can enable Secure Sockets Layer (SSL) encrypted communication using the SSLCIPHERSUITE property. Set this property to a valid CipherSuite provided by your JSSE provider; it must match the CipherSpec named on the SVRCONN channel named by the CHANNEL property.

However, CipherSpecs (as specified on the SVRCONN channel) and CipherSuites (as specified on ConnectionFactory objects) use different naming schemes to represent the same SSL encryption algorithms. If a recognized CipherSpec name is specified on the SSLCIPHERSUITE property, JMSAdmin issues a warning and maps the CipherSpec to its equivalent CipherSuite. See SSL CipherSpecs and CipherSuites in JMS for a list of CipherSpecs recognized by WebSphere MQ and JMSAdmin.

If you require a connection to use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS), set the SSLFIPSREQUIRED property of the connection factory to YES. The default value of this property is NO, which means that a connection can use any supported CipherSuite. The property is ignored if SSLCIPHERSUITE is not set.

The SSLPEERNAME matches the format of the SSLPEER parameter, which can be set on channel definitions. It is a list of attribute name and value pairs separated by commas or semicolons. For example:

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

The set of names and values makes up a *distinguished name*. For more details about distinguished names and their use with WebSphere MQ, see .

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is not case-sensitive.

If SSLPEERNAME is not set, no such checking is performed. SSLPEERNAME is ignored if SSLCIPHERSUITE is not set.

The SSLCRL property specifies zero or more CRL (Certificate Revocation List) servers. Use of this property requires a JVM at Java 2 v1.4. This is a space-delimited list of entries of the form:

```
ldap://hostname:[port]
```

optionally followed by a single /. If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. See for more about CRL security.

If SSLCRL is not set, no such checking is performed. SSLCRL is ignored if SSLCIPHERSUITE is not set.

The SSLRESETCOUNT property represents the total number of bytes sent and received by a connection before the secret key that is used for encryption is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by WebSphere MQ classes for JMS.

For example, to configure a ConnectionFactory object that can be used to create a connection over an SSL enabled MQI channel with a secret key that is renegotiated after 4 MB of data have flowed, issue the following command to JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

If the value of SSLRESETCOUNT is zero, which is the default value, the secret key is never renegotiated. The SSLRESETCOUNT property is ignored if SSLCIPHERSUITE is not set.

Rules for selecting the WebSphere MQ messaging provider mode

The WebSphere MQ messaging provider has two modes of operation: WebSphere MQ messaging provider normal mode, and WebSphere MQ messaging provider migration mode. You can select which mode a JMS application uses to publish and subscribe.

The WebSphere MQ messaging provider normal mode uses all the features of a MQ queue manager to implement JMS. This mode is used only to connect to a WebSphere MQ queue manager and can connect to queue managers in either client or bindings mode. This mode is optimized to use the new function.

If you are not using WebSphere MQ Real-Time Transport, the mode of operation used is determined primarily by the **PROVIDERVERSION** property of the connection factory. If you cannot change the connection factory that you are using, you can use the `com.ibm.msg.client.wmq.overrideProviderVersion` property to override any setting on the connection factory. This override applies to all connection factories in the JVM but the actual connection factory objects are not modified.

You can set **PROVIDERVERSION** to the possible values: `7`, `6`, or *unspecified*. However, **PROVIDERVERSION** can be a string in any one of the following formats:

- V.R.M.F
- V.R.M
- V.R
- V

where V, R, M and F are integer values greater than or equal to zero.

7 - Normal mode
Uses the WebSphere MQ messaging provider normal mode.

If you set **PROVIDERVERSION** to 8 only the WebSphere MQ messaging provider normal mode of operation is available. If the queue manager specified in the connection factory settings is not a Version 7.0.1, or later, queue manager, the createConnection method fails with an exception.

The WebSphere MQ messaging provider normal mode uses the sharing conversations feature and the number of conversations that can be shared is controlled by the **SHARECNV()** property on the server connection channel. If this property is set to 0, you cannot use WebSphere MQ messaging provider normal mode and the createConnection method fails with an exception.

6 - Migration mode
Uses the WebSphere MQ messaging provider migration mode.

The WebSphere MQ classes for JMS use the features and algorithms supplied with WebSphere MQ version 6.0. If you want to connect to WebSphere Event Broker version 6.0 or WebSphere Message Broker version 6.0 or 6.1 using WebSphere MQ Enterprise Transport version 6.0, you must use this mode. You can connect to a WebSphere MQ version 7.0.1 queue manager using this mode, but none of the new features of a version 7.0.1 queue manager are used, for example, read ahead or streaming. If you have a WebSphere MQ version 7.0.1 client connecting to a WebSphere MQ version 7.0.1 queue manager on a distributed platform or a WebSphere MQ version 7.0.1 queue manager on z/OS, then the message selection is done by the queue manager rather than on the client system.

unspecified
The **PROVIDERVERSION** property is set to *unspecified* by default.

A connection factory that was created with a previous version of WebSphere MQ classes for JMS in JNDI takes this value when the connection factory is used with the new version of WebSphere MQ classes for JMS. The following algorithm is used to determine which mode of operation is used. This algorithm is used when the createConnection method is called and uses other aspects of the connection factory to determine if WebSphere MQ messaging provider normal mode or WebSphere MQ messaging provider migration mode is required.

1. First, an attempt to use WebSphere MQ messaging provider normal mode is made.
2. If the queue manager connected is not WebSphere MQ version 7.0.1, or later, the connection is closed and WebSphere MQ messaging provider migration mode is used instead.
3. If the **SHARECNV** property on the server connection channel is set to 0, the connection is closed and WebSphere MQ messaging provider migration mode is used instead.
4. If **BROKERVER** is set to V1 or the new default *unspecified* value, WebSphere MQ messaging provider normal mode continues to be used, and therefore any publish/subscribe operations use the new WebSphere MQ version 7.0.1, or later, features.

   See ALTER QMGR for information about the PSMODE parameter of the ALTER QMGR command for further information on compatibility.
5. If **BROKERVER** is set to V2 the action taken depends on the value of **BROKERQMGR** :
   - If the **BROKERQMGR** is blank:

     If the queue specified by the **BROKERCONQ** property can be opened for output (that is, MQOPEN for output succeeds) and **PSMODE** on the queue manager is set to COMPAT or DISABLED, then WebSphere MQ messaging provider migration mode is used.

- If the queue specified by the **BROKERCONQ** property cannot be opened for output, or the **PSMODE** attribute is set to ENABLED:

  WebSphere MQ messaging provider normal mode is used.
- If **BROKERQMGR** is non-blank :

  WebSphere MQ messaging provider mode is used.

Related information:

When to use PROVIDERVERSION

BROKERQMGR

BROKERCONQ

PSMODE

When to use PROVIDERVERSION

In two cases you must override the default selection of **PROVIDERVERSION** for the WebSphere MQ classes for JMS to work correctly.

There are two scenarios where you cannot use the algorithm described in Rules for selecting the WebSphere MQ messaging provider mode; consider using **PROVIDERVERSION** in these scenarios.

1. If WebSphere Event Broker or WebSphere Message Broker is in compatibility mode, you must specify **PROVIDERVERSION** for them to work correctly.
2. If you are using WebSphere Application Server Version 6.0.1, WebSphere Application Server Version 6.0.2, or WebSphere Application Server Version 6.1, connection factories are defined using the WebSphere Application Server administrative console.

   In WebSphere Application Server the default value of the **BROKERVER** property on a connection factory is V2. The default **BROKERVER** property for connection factories created by using **JMSAdmin** or WebSphere MQ Explorer is V1. This property is now "unspecified" in WebSphere MQ.

If **BROKERVER** is set to V2 (either because it was created by WebSphere Application Server or the connection factory has been used for publish/subscribe before) and has an existing queue manager that has a **BROKERCONQ** defined (because it has been used for publish/subscribe messaging before), the WebSphere MQ messaging provider migration mode is used.

However, if you want the application to use peer-to-peer communication and the application is using an existing queue manager that has ever done publish/subscribe, and has a connection factory with **BROKERVER** set to 2 (if the connection factory was created in WebSphere Application Server this is the default), the WebSphere MQ messaging provider migration mode is used. Using WebSphere MQ messaging provider migration mode in this case is unnecessary; use WebSphere MQ messaging provider normal mode instead. You can use one of the following methods to work around this:

- Set **BROKERVER** to 1 or unspecified. This is dependent on your application.
- Set **PROVIDERVERSION** to 7, which is a custom property in WebSphere Application Server Version 6.1. The option to set custom properties in WebSphere Application Server Version 6.1 and later is not currently documented in the WebSphere Application Server product documentation.

  Alternatively, use the client configuration property, or modify the queue manager connected so it does not have the **BROKERCONQ**, or make the queue unusable.

IBM WebSphere MQ Telemetry Reference

Information about programming MQTT clients

MQ Telemetry Transport format and protocol

The MQ Telemetry Transport (MQTT) is a lightweight publish/subscribe protocol flowing over TCP/IP to connect large numbers of remote sensors and control devices. MQTT is used by specialized applications on small footprint devices that must tolerate low bandwidth and unreliable communication. You can write your own clients to use the published protocol, or use one of the clients supplied with the installation of IBM WebSphere MQ Telemetry. There are additional MQTT clients available as SupportPacs, and from business partners.

IBM WebSphere MQ Telemetry uses version 3.1 of the MQ Telemetry Transport (MQTT) protocol. IBM publishes the protocol specification at http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

If you have obtained an MQTT client from a source other then an installation of IBM WebSphere MQ, check the version of the MQTT protocol supported by the client.

Currently, clients from sources other than IBM WebSphere MQ Telemetry typically support a different level of the MQTT protocol and do not work correctly with the IBM WebSphere MQ Telemetry service. For these clients, a thin conversion layer is required that converts the clients to MQTT v3.1. Check with the source of your client if the conversion layer is available as an update to the client you intend to use.

WebSphere MQ Telemetry daemon for devices reference information

Reference information for configuring the WebSphere MQ Telemetry daemon for devices.

WebSphere MQ Telemetry daemon for devices configuration file

Use the daemon configuration file to set WebSphere MQ Telemetry daemon for devices configuration parameters. The configuration file contains three types of parameters that control the daemon: global, bridge, and listener parameters.

Daemon configuration file

WebSphere MQ Telemetry daemon for devices configuration options are selected by entries in the daemon configuration file. The default configuration file is named `amqtdd.cfg`. It is in the same directory as the daemon executable program.

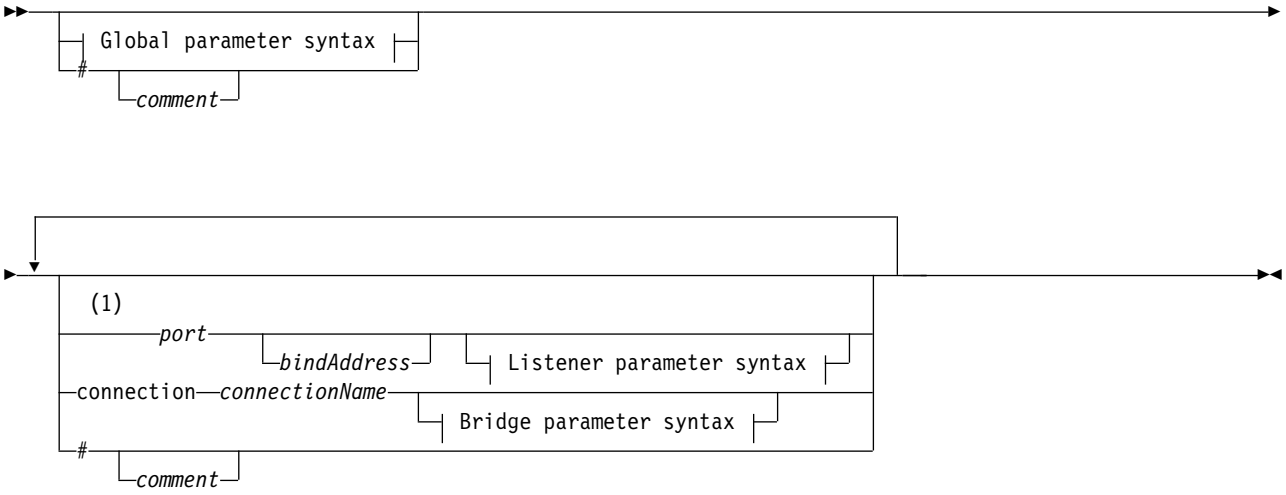
Specify a different configuration file by passing the path and file name as a single parameter when you start the daemon. For example, if the configuration file is called `testdaemon.cfg`, enter the following command to start the daemon:

```
./amqtdd testdaemon.cfg
```

When started, the daemon checks for the existence of the configuration file. If the file does not exist, the daemon runs with default settings.

You can change some of the configuration options while the WebSphere MQ Telemetry daemon for devices is running. Place the updates in a file named `amqtdd.upd`. See Modifying daemon configuration while it is running for the complete list of the commands and options that you can place in `amqtdd.upd`.

Configuration file syntax



Global parameter syntax:

Bridge parameter syntax:

Listener parameter syntax:

Notes:

1 A default listener exists on port. port is a global parameter and defaults to 1883

2 See "Global parameters syntax" on page 2809.

3 See Bridge parameters syntax.

4 See Listener parameters syntax.

The configuration file is a text file. Type each configuration parameter in the configuration file on a single line. You can format the file with spaces and tabs anywhere on a line.

Configuration file parameters

Bridge parameters

Bridge parameters control how this daemon connects to another publish/subscribe broker using the MQTT v3 protocol; see "Bridge parameters" on page 2815.

Bridge parameters must follow any global parameters. All bridge parameters for each connection must be on consecutive lines.

Note: The term bridge is used to describe the bridge component of the daemon. The bridge component makes connections to other brokers using the MQTT V3 protocol and propagates publications from broker to broker; see WebSphere MQ Telemetry daemon for devices bridgesWebSphere MQ Telemetry daemon for devices bridges. A connection is a instance of the bridge that connects to a specific broker. Examples of connections would be a connection to WebSphere MQ using a WebSphere MQ Telemetry channel, or a connection to another daemon.

connection *connectionName*

The name of the connection. The name must be alphanumeric; for example, `connection1`. A connection connects the daemon to a queue manager using a WebSphere MQ telemetry channel or to another daemon using a listener; see "WebSphere MQ Telemetry daemon for devices listener parameters" on page 2818.

connectionName is combined with the system *hostname* to create a `ClientIdentifier`. `ClientIdentifier` identifies the bridge to the listener or telemetry channel it connects to. The bridge is an MQTT v3 client.

Connection indicates the start of a bridge connection section in the configuration file and must follow all the global parameters. Listener sections and bridge sections can occur in any order.

Global parameters

Global parameters control the overall operation of the daemon; see "Global parameters" on page 2811. Global parameters must precede any listener or bridge parameters.

listener *portNumber*|**default** *bindAddress*

Creates a new listener with the specified *portNumber* and an optional local *bindAddress*; see bind_address. The listener connects MQTT clients to the daemon.

listener indicates the start of a listener section in the configuration file and must follow all the global settings. Listener sections and bridge sections can occur in any order.

Listener parameters

Listener parameters control how MQTT clients and other daemons connect to this WebSphere MQ daemon for devices; see "Listener parameters" on page 2819. Listener parameters must follow any global parameters. All listener parameters for each listener must be on consecutive lines.

 # comment

Comments can be placed on any line in the file, by placing a # as the first nonwhite-space character on the line. Trailing comments on a line are not supported.

Example configuration file

```
# Sample configuration
# Daemon listens on port 1882 with persistence in /tmp
port 1882
persistence_location /tmp/
retained_persistence true
```

WebSphere MQ Telemetry daemon for devices global parameters

Set global parameters in the daemon configuration file to control the daemon.

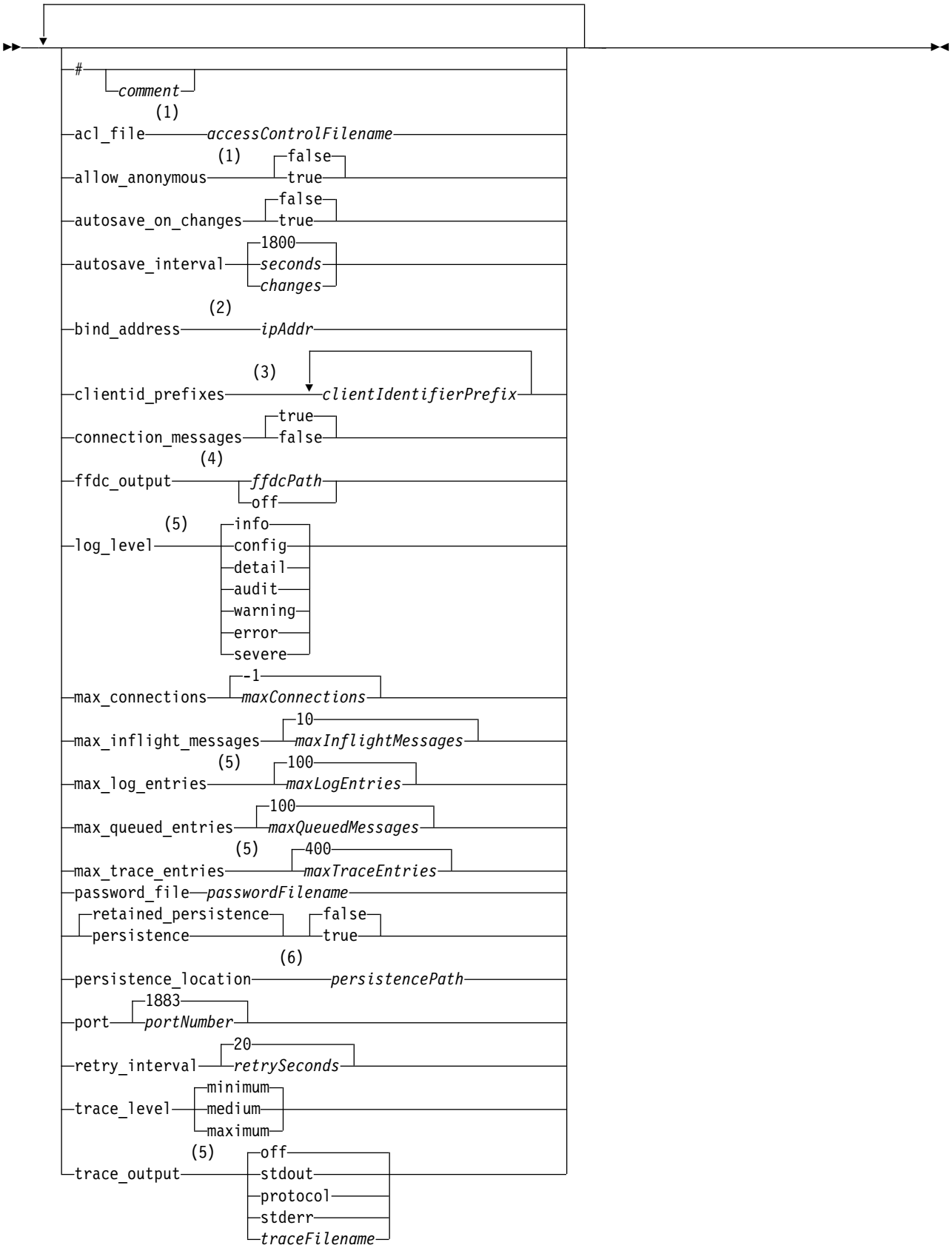
Global parameters syntax

Global parameter settings must precede any bridge or listener sections in the configuration file.

The name and format of the configuration file are described in "WebSphere MQ Telemetry daemon for devices configuration file" on page 2807.

You can modify some of the parameters, while the daemon is running, by placing updates in the `amqtdd.upd` file; see Modifying the daemon while it is running.

Global parameters syntax



Notes:

1 Only allowed if *passwordFilename* specified.

2 The default is connections from all network interfaces are allowed.

3 The default is any client identifiers allowed.

4 The default path is *persistencePath*.

5 Update this parameter while WebSphere MQ Telemetry daemon for devices is running by placing it in the `amqtdd.upd` file.

6 The default path is the installation directory for the WebSphere MQ Telemetry daemon for devices.

Global parameters

Global parameters control the overall operation of the daemon.

 # comment

Comments can be placed on any line in the file, by placing a # as the first nonwhite-space character on the line. Trailing comments on a line are not supported.

acl_file *accessControlFilename*

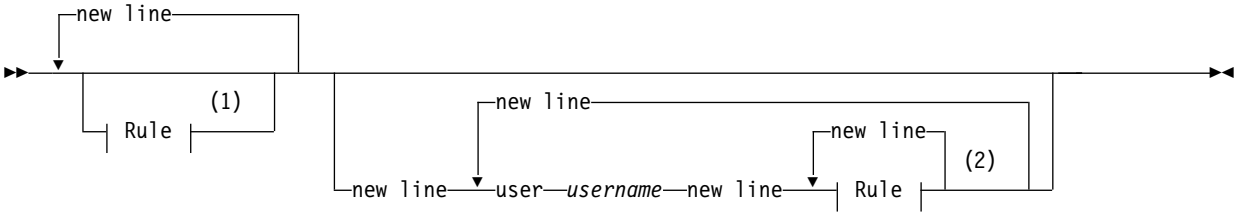
accessControlFilename is the name of a file containing access control rules. The default is not to provide an access control file, and not to apply any access control. Access control is turned on only if `password_file` and *accessControlFilename* are specified. If access control is turned on, the default is to restrict access to every topic. Access is granted to topics by rules in the access control file.

The file is in plain text, with one access control rule per line. The first set of rules is universal, and apply to all users, including anonymous users. After the universal rules, there are sets of rules for any user in the password definition file.

Each rule is a permission, followed by a topic string that can contain wildcards that identify the topics to which the permission is applied. The effect of the rules is cumulative. That is, the daemon starts with no one permitted access any topic. It applies each rule to add to the topics each user is permitted to read and write.

The file is organized as follows:

Access control file syntax



Rule:

Permission:

Notes:

1 Universal rules

2 User-specific rules

3 read/write

The access control file has the following parameters:

permission topicString

Add read or write, or read and write permission to topics that match *topicString*. The rules apply either to all users, or in the user sections of the file, to individual users. The effect of the rules is additive. The rules extend the set of topics a user is allowed to read and write.

Rules that provide full, or read access, cannot use the + wildcard. Write-only rules can use the + wildcard.

Topics in the access control list file must include topic prefixes applied by the use of mount points.

user *username*

The following rules apply to the user in the password file with the user ID, *username*.

allow_anonymous true|false

allow_anonymous is applicable only if `password_file` has been specified. Set allow_anonymous to true to allow clients to connect without providing authentication information. Set allow_anonymous to false to force clients to provide authentication information. See Authentication of clients.

autosave_on_changes true|false

Set autosave_on_changes to change how the value of autosave_interval is used. Set autosave_on_changes to true to trigger an autosave when the number of changes reaches *autosaveChanges*. Set autosave_on_changes to false to trigger an autosave when the number of seconds since the last autosave reaches *autosaveSeconds*.

autosave_interval *autosaveSeconds*|*autosaveChanges*|**1800**

autosave_interval is the length of the autosave interval either in seconds or the number of changes, depending on the autosave_on_changes setting. `0` means no autosave. See Saving retained messages and subscriptions.

bind_address *ipAddr*

The default bind_address value is for the daemon to allow connections from all network interfaces

ipAddr is the local IP address to bind to for the default listener. Use bind_address if the host system has multiple network cards and you want to limit access to be from one network. Specify *ipAddr* as `127.0.0.1` to restrict client connections only to connections from the same workstation as the daemon.

clientid_prefixes *clientIdentifierPrefix*

clientid_prefixes is a list of prefixes to restrict the clients that are allowed to connect to the daemon. Only clients with client identifiers that start with *clientIdentifierPrefix* are allowed to connect. Any other connections are rejected. For example, setting *clientIdentifierPrefix* to `test_` allows only clients with client identifiers such as `test_1` and `test_connection` to connect.

connection_messages true|false

Set connection_messages to true to log client connection and disconnection messages. Set connection_messages to false to turn logging of connection messages off.

ffdc_output *ffdcPath*|**off|Persistence_location**

The default value of ffdc_output is persistencePath.

ffdcPath is the directory path, excluding the file name, used to store FFDC files. The prefix must include the trailing directory separator, / or \.

The value off turns off FFDC writing altogether. Turning off FFDC writing makes problem determination difficult.

log_level config|detail|info|audit|warning|error|severe

log_level is the level of log output required. The log levels are listed in order of increasing importance.

Log messages are written to `stdout` and to the `$SYS/broker/log` topic.

max_connections *maxConnections*|**-1**

The default value of max_connections is `-1`, no limit.

maxConnections is the maximum number of active clients that can connect to the default port. See Listener settings to set this parameter for other ports.

max_inflight_messages *maxInflightMessages*|**10**

maxInflightMessages is the maximum number of QoS=1 or QoS=2 outbound messages that are being acknowledged or sent again for a client; see Qualities of service provided by an MQ Telemetry Transport client.

max_log_entries *maxLogEntries*|**100**

maxLogEntries is the maximum number of log entries remembered for retrieval by the **trace_dump** command or in an FFDC.

max_queued_entries *maxQueuedMessages*|**100**

maxQueuedMessages is the maximum number of QoS=1 or QoS=2 messages that can be queued for delivery to each client; see Qualities of service provided by an MQ Telemetry Transport client.

Note: If the queue of messages for a client fills up, any subsequent messages for that client are discarded and are not delivered to that client. When the queue is able to accept messages again, normal message delivery resumes.

max_trace_entries *maxTraceEntries*|**400**

maxTraceEntries is the maximum number of trace entries remembered for retrieval by the **trace_dump** command or in an FFDC.

password_file *passwordFilename*

The default, having no password file, is not to apply authentication.

passwordFilename is the name of a file containing user name and password authentication information. The file is in plain text, with one password definition per line. Each definition has the following format:

username:*password*

persistence|retained_persistence true|false

Set retained_persistence to true to save retained publications and durable subscriptions when the daemon is shut down and restored when the daemon restarts. Set retained_persistence to false to discard retained messages and subscriptions. See Saving retained messages and subscriptions.

Note: Persistence and retained_persistence are synonyms. Use retained_persistence in preference to persistence.

persistence_location *persistencePath*

The default persistence_location is the directory in which the daemon is installed.

persistencePath is the directory path to store retained messages and durable subscriptions. The path must include the trailing directory separator, / or \ and does not include a file name.

port *portNumber*|**1883**

The default listener uses *portNumber* to listen for MQTT client connections.

retry_interval *retrySeconds*|**20**

retrySeconds is the number of seconds before the daemon tries to send an unacknowledged message with at least once or at most once quality of service again.

trace_level **minimum**|**medium**|**maximum**

trace_level is the level of trace taken and stored in an internal buffer.

trace_output **off**|**stdout**|**stderr**|**protocol**|*tracePath*

trace_output is the destination to write trace entries as they occur. It also controls whether a full trace or just a message trace is taken.

Tracing continues indefinitely until explicitly turned off and results in large files.

The **protocol** setting writes an entry for every MQTT message sent to or received from a client to stdout.

The stdout, stderr and *tracePath* settings write a complete trace to the specified destination.

tracePath is either a path, or a file name, relative to the working directory.

IBM WebSphere MQ Telemetry daemon for devices bridge parameters

Configure a IBM WebSphere MQ Telemetry daemon for devices bridge connection by setting bridge parameters in the daemon configuration file.

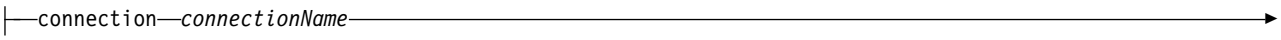
See WebSphere MQ Telemetry daemon for devices bridgesWebSphere MQ Telemetry daemon for devices bridges for an explanation and examples showing how a bridge connection propagates publications to and from the IBM WebSphere MQ Telemetry daemon for devices.

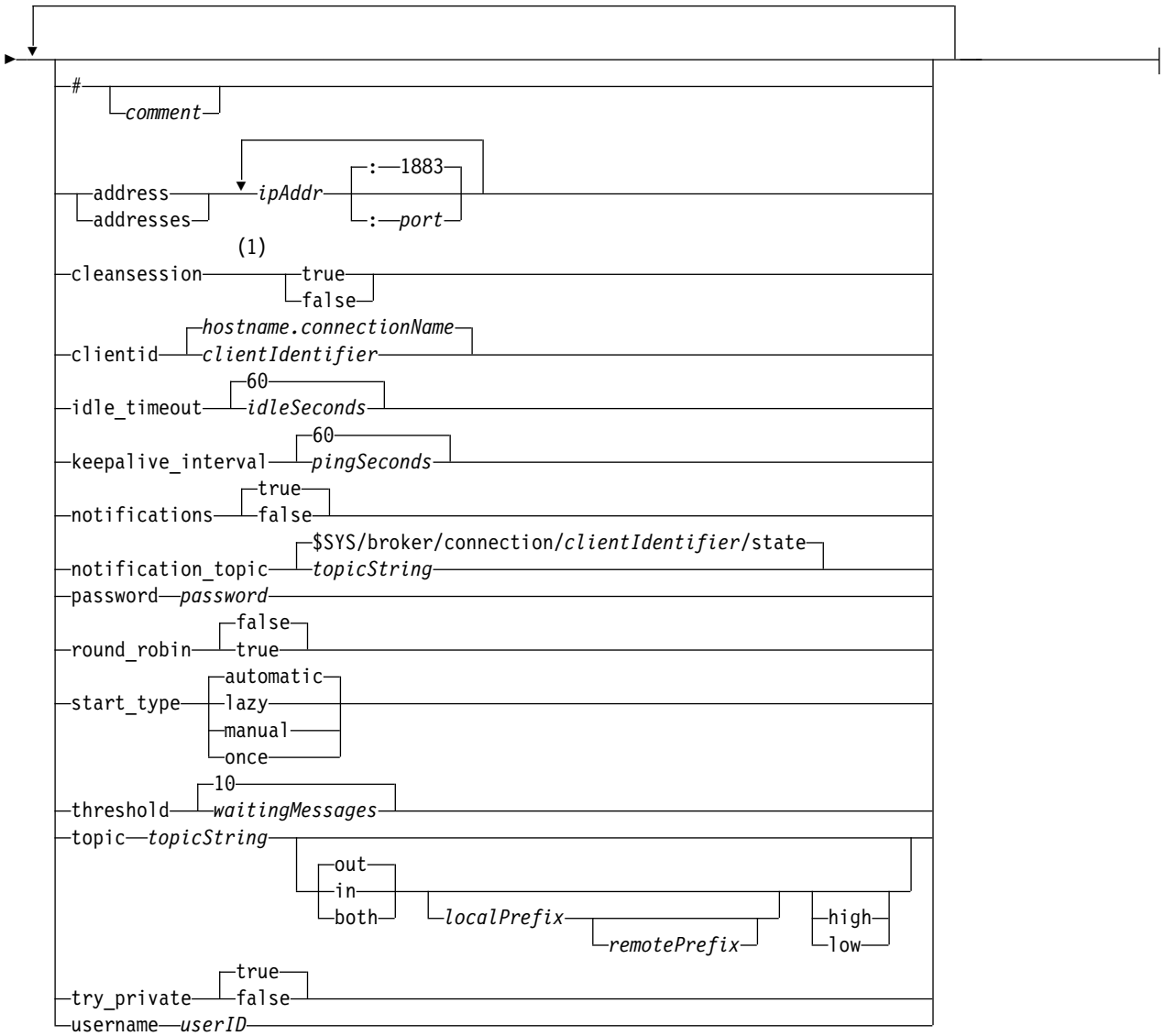
Bridge parameters syntax

Each bridge section of the configuration file starts with a connection parameter, see "WebSphere MQ Telemetry daemon for devices configuration file" on page 2807. The parameters specific to a particular connection immediately follow the connection entry.

The only parameters allowed in the file following a bridge section are parameters belonging to listener sections or additional bridge sections.

Connection:





Notes:

1 If the number of addresses is greater than one, cleansession is true by default, otherwise it is false.

Bridge parameters

 # comment
Comments can be placed on any line in the file, by placing a # as the first nonwhite-space character on the line. Trailing comments on a line are not supported.

address|addresses *ipAddr*:*port*|**1883**
addresses[10] is a list of TCP/IP socket addresses to which the daemon attempts to make a bridge connection. By default, the first address in the list is treated as the primary server; see round_robin.

Use multiple addresses with WebSphere MQ Telemetry in the following configurations;

Multiple queue managers and multiple network addresses.
The list of *ipAddr* connects to telemetry channels on different queue managers. Set

10. Address and Addresses are synonyms. Use either.

round_robin to false if one network address is preferred. Make this the first address in the list. Set cleansession to true. If cleansession is set to false, unpredictable behavior, including lost publications and subscriptions results.

Single multi-instance queue manager
Provide two addresses; the first address is the active queue manager instance and the second address is the standby. Set round_robin to true and cleansession to false.

Single queue manager and multiple network addresses
In this configuration the list of IP addresses all connect to the same queue manager through different network paths. The queue manager is configured with multiple telemetry channels listening to different socket addresses. You might configure the server in this way to introduce redundancy into the network connectivity, or to spread the load of many client connections over multiple network adapters. Set round_robin to false if one network address is preferred. Make this the first address in the list. Set cleansession to false.

See Availability of IBM WebSphere MQ Telemetry daemon for devices bridge connections for more information about using multiple addresses.

cleansession true|false

The default value of cleansession is true if the number of addresses is greater than one, otherwise it is false.

cleansession controls session state when the daemon connects, disconnects, and reconnects. Session state includes subscriptions and queued messages.

Set cleansession to true to discard session state when connecting and disconnecting. Set cleansession to false to save state on disconnecting and restore state on connecting, if possible.

Note: Do not set cleansession to false if addresses lists multiple IP addresses, and the IP addresses connect to telemetry channels hosted by different queue managers, or to different telemetry daemons. Session state is not transferred between queue managers or daemons. Trying to restart an existing session on a different queue manager or daemon results in a new session being started. In-doubt messages are lost, and subscriptions might not behave as expected.

clientid *clientIdentifier*|*hostname.connectionName*

The default *clientIdentifier* is constructed from concatenating the daemon host name with *connectionName*. The host name is truncated after the first `'.'` character or 14 characters, whichever is fewer. The combination is truncated at 23 characters if it is longer than 23 characters.

clientid is passed to the remote server when connecting.

clientid must only contain characters from the range: `A-Z, a-z, 0-9, './_%`.

connection *connectionName*

The name of the connection. The name must be alphanumeric; for example, `connection1`. A connection connects the daemon to a queue manager using a WebSphere MQ telemetry channel or to another daemon using a listener; see "WebSphere MQ Telemetry daemon for devices listener parameters" on page 2818.

connectionName is combined with the system *hostname* to create a `ClientIdentifier`. `ClientIdentifier` identifies the bridge to the listener or telemetry channel it connects to. The bridge is an MQTT v3 client.

Connection indicates the start of a bridge connection section in the configuration file and must follow all the global parameters. Listener sections and bridge sections can occur in any order.

idle_timeout *idleSeconds*|**60**

Set *idleSeconds* to the number of seconds to elapse before the connection is closed.

keepalive_interval *pingSeconds*|**60**

Set *pingSeconds* to the number of seconds between sending MQTT ping requests to the remote system when there has been no other traffic. The minimum value is 5.

notifications true|false

Set notifications to true to switch on bridge connection notifications. Set notifications to false to switch off bridge notifications.

Notifications are retained messages published at both ends of the bridge published to a specially defined topic; see notification_topic.

The notification publication contains a single character indicating the status of the bridge connection. The status is either `1`, connected, or `0`, disconnected.

The status of a bridge connection can be checked at any time.

notification_topic *topicString*|**$SYS/broker/connection/***clientIdentifier***/state**

The default notification_topic is **$SYS/broker/connection/***clientIdentifier***/state**. The default topic includes the *clientIdentifier* of the bridge connection.

Set *topicString* to an alternative topic, if you want to use a different topic to track connection status.

Connection notification messages, with the value `1`, connected, or `0`, disconnected, are published to this topic.

Note: The default *topicString* contains the prefix `$SYS`. Subscribe to topics beginning with `$SYS` by defining a topic filter beginning with `$SYS`. The topic filter `#`, subscribe to everything, does not subscribe to topics beginning with `$SYS` on the daemon. Think of `$SYS` as defining a special system topic space distinct from the application topic space.

password *password*

The default is not to set *password*.

Sets a *password*, which is used in combination with *userID* to authenticate the connection to the remote broker. If the remote connection is to a WebSphere MQ telemetry channel, *userID* is authenticated using JAAS.

round_robin true|false

Set round_robin to true to connect to each address in the addresses list until it is successful. The daemon tries each address in turn starting with the first address, the primary server.

Set round_robin to false to force the daemon to connect to the primary server whenever it is available.

If the primary server is unavailable, the daemon tries each address in turn until it makes a connection. It keeps trying to connect to the primary server in the background. As soon as the primary server becomes available again, the daemon reconnects to it, dropping the connection it is currently using.

start_type automatic|lazy|once|manual

Set start_type to automatic to keep the bridge connected. The connection opens as soon as the daemon starts. If the connection fails, the daemon restarts it after about 20 seconds.

Set to start_type to lazy to reduce network usage and costs. The connection starts when the number of messages waiting reaches *waitingMessages*. The connection is closed when the bridge has been idle for *idleSeconds*.

Set to start_type to manual to start and stop the bridge using start and stop commands; see Modifying daemon configuration while it is running.

Set start_type to once to connect the bridge when the daemon is started and to delete it if it is stopped or disconnected. If start_type is set to once and the bridge is stopped manually, or disconnected due to an error, the bridge cannot be restarted until the daemon is restarted.

threshold *waitingMessages*

If start_type is lazy, the connection starts when the number of queued messages reaches *waitingMessages*.

topic *topicString* **in|out|both** *local_prefix remote_prefix* **high|low**

The value of the topic parameter can comprise up to five parts:

The first part, *topicString* can be prefixed by an additional topic string: *localPrefix*, or *remotePrefix*. Unlike *topicString*, *localPrefix* and *remotePrefix* must not contain wildcards. *localPrefix* and *remotePrefix* usually end with a / character, to align with topic hierarchies at each end of the bridge.

The second part, which takes the values, **in** | **out** | **both**, is called the direction parameter. out is the default setting.

If the direction is out, then the bridge connection subscribes to publications at the local daemon using the topic filter *localPrefix*||*topicString*. The publications that are selected are published to the remotely attached broker with the topic string *remotePrefix*||*topicString*.

If the direction is in, then the bridge connection subscribes to publications at the remote broker using the topic filter *remotePrefix*||*topicString*. The publications that are selected are published to the local daemon with the topic string *localPrefix*||*topicString*.

If the direction is both, then the result is the same as having two topic settings, one set to in and one set to out. Only use the both setting when the brokers have a publication loop detection mechanism. A loop detection mechanism stops a publication entering a perpetual loop. There is no loop detection for a bridge is connected to a WebSphere MQ telemetry channel; see try_private.

The optional fifth part is priority, which signifies the priority given to messages sent or received through the bridge, where the topic matches *topicString*. The priority applies only within the WebSphere MQ Telemetry daemon for devices. You can specify a value of **high** or **low**. If you do not specify a value, the priority is between **high** and **low**. When messages are queued for a client or bridge connection, the messages with the highest priority are sent first. When multiple topics are applied to one bridge connection, the priority applied to a message is the one from the first matching topic. Therefore it is important to consider the order in which the priority value is set in the configuration file.

See Example topic settings for the bridgeWebSphere MQ Telemetry daemon for devices bridges for examples of using the topic parameter.

try_private **true**|**false**

Set **try_private** to check whether the remote broker is another instance of the daemon. If the remote broker is another WebSphere MQ Telemetry daemon for devices, and try_private is set to true, then publication loops between a pair of daemons are detected. Loops involving more complex topologies might not be detected.

username *userID*

The default is not to set *userID*.

Sets a *userID*, which is used in combination with *password* to authenticate the connection to the remote broker. If the remote connection is to a WebSphere MQ telemetry channel, *userID* is authenticated using JAAS.

userID is used for access control if the remote connection is to a daemon. If the remote connection is to a telemetry channel, you have the choice of using *userID* for authorization, or using another identification; see MQTT client identification, authorization, and authentication.

WebSphere MQ Telemetry daemon for devices listener parameters

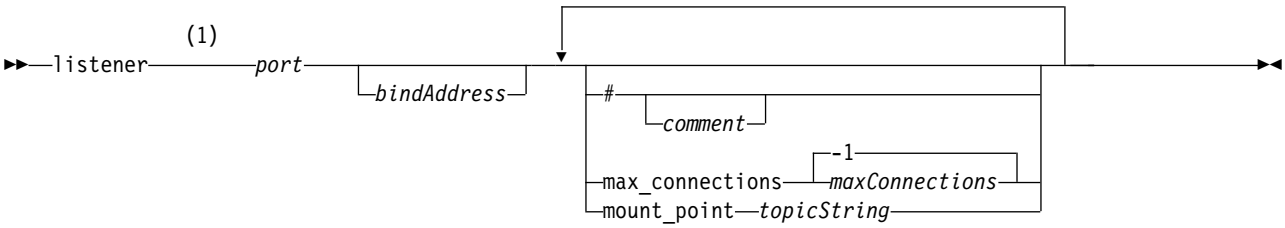
Configure a WebSphere MQ daemon for devices listener by setting listener parameters in the daemon configuration file. MQTT clients and other daemons can connect to a listener and publish and subscribe to topics at the daemon.

Listener parameters syntax

Each listener section of the configuration file starts with a listener parameter, see "WebSphere MQ Telemetry daemon for devices configuration file" on page 2807. The parameters specific to a particular listener immediately follow the listener entry.

The only parameters allowed in the file following a listener section are bridge sections or additional listener sections.

Listener parameters syntax



Notes:

1 A default listener exists on port. port is a global parameter and defaults to 1883

Listener parameters

Configure a listener using the following parameters:

 # comment
Comments can be placed on any line in the file, by placing a # as the first nonwhite-space character on the line. Trailing comments on a line are not supported.

listener *portNumber*|**default** *bindAddress*

Creates a new listener with the specified *portNumber* and an optional local *bindAddress*; see bind_address. The listener connects MQTT clients to the daemon.

listener indicates the start of a listener section in the configuration file and must follow all the global settings. Listener sections and bridge sections can occur in any order.

max_connections *maxConnections*|**-1**

The default value of max_connections is -1, no limit.

Set *maxConnections* to the maximum number of active clients that are allowed to be connected to the port simultaneously.

You can set the global parameter, max_connections to set *maxConnections* for the default port.

mount_point *topicString*

A string that is prefixed to all topic strings published by and subscribed to by clients connecting to this listener. This can be used to ensure clients on different listeners cannot interfere with each other; see Mount points.

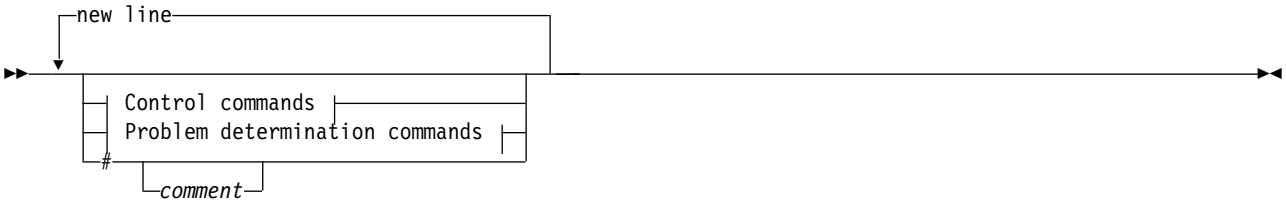
WebSphere MQ Telemetry daemon for devices command file

Use the daemon command file to modify the behavior of a running daemon. You can start and stop a bridge connection, stop the daemon, clear retained publications, and do problem determination.

Command file syntax

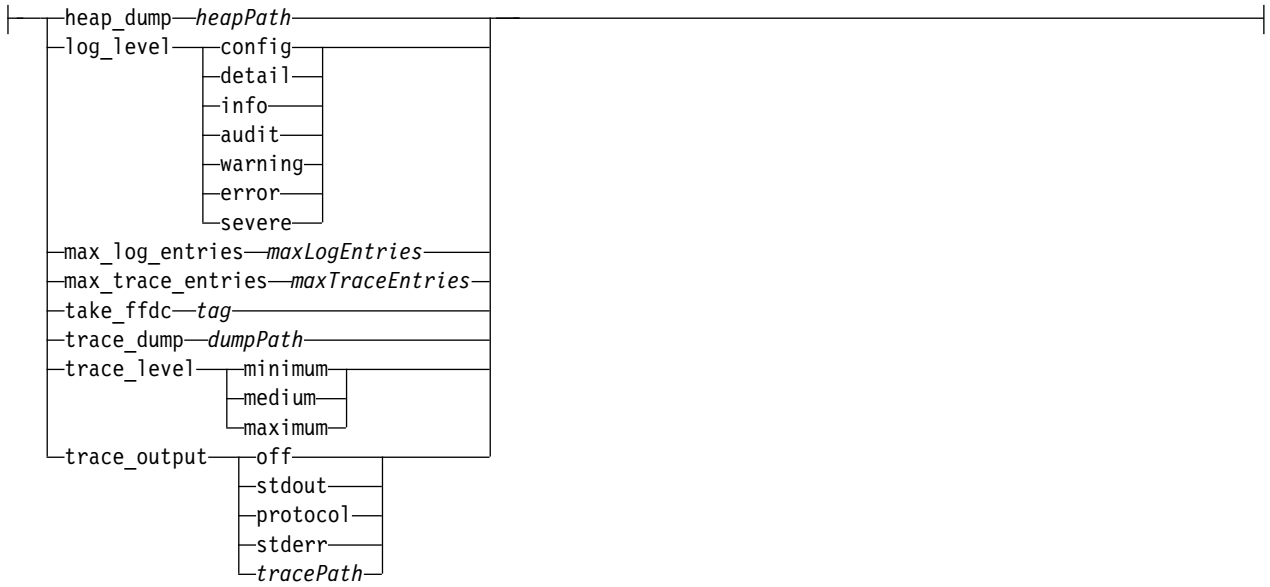
Place commands in the command file, `amqtdd.upd`. Every 5 seconds the daemon runs the commands in the file, and deletes the file.

Each command is a separate line in the command file. The commands are acted upon, in order, line by line. Unrecognized commands are written to the command window from which the daemon was started.

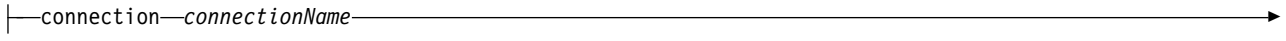


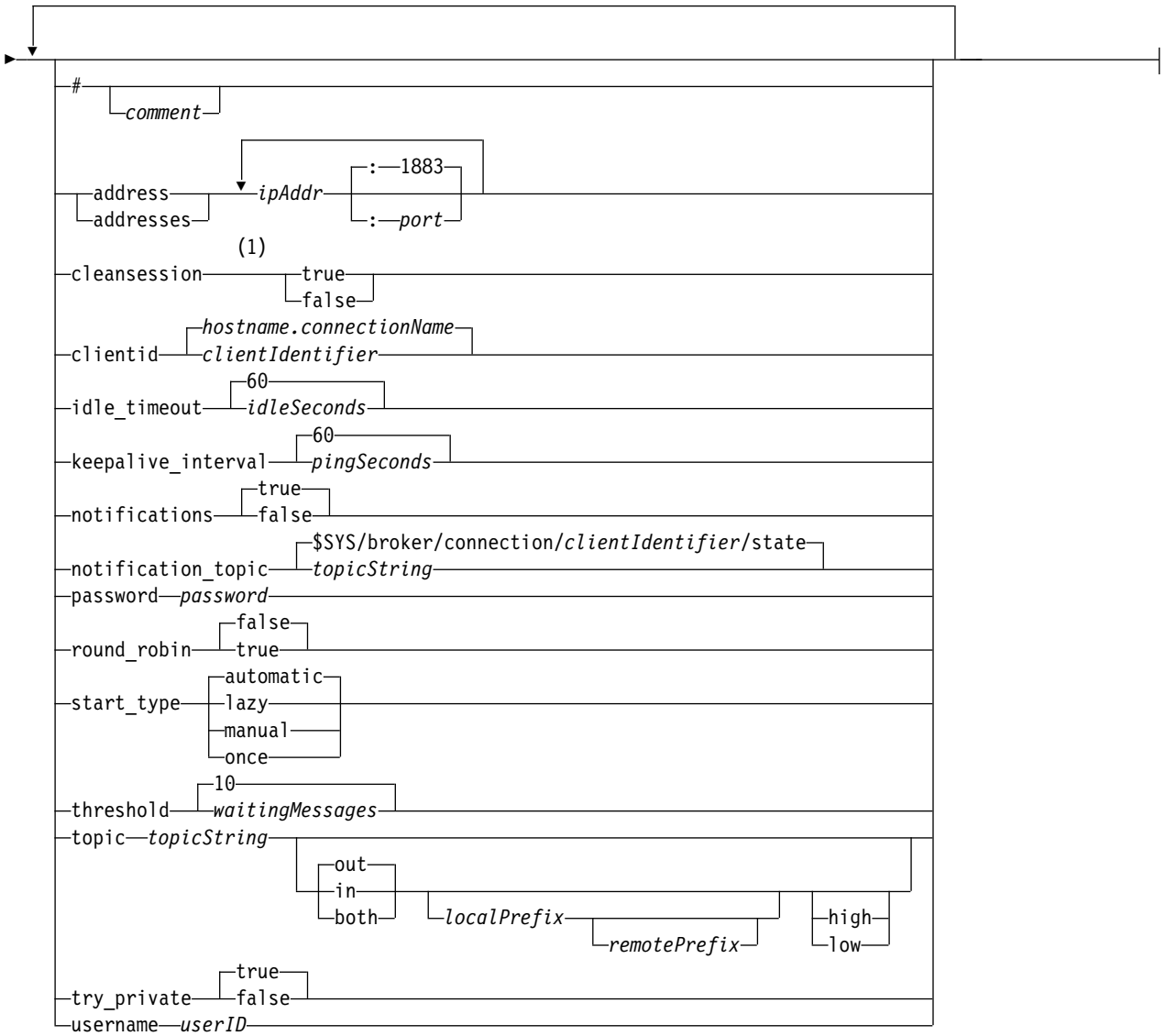
Control commands:

Problem determination commands:



Connection:





Notes:

1 If the number of addresses is greater than one, cleansession is true by default, otherwise it is false.

Control commands

clear_retained *topicString*

Remove retained messages for any topics that match *topicString*. *topicString* can contain wildcards.

Connection

See "Bridge parameters" on page 2815.

delete_connection *connectionName*

Delete the bridge connection *connectionName*. If the connection is running, it is stopped first.

start_connection *connectionName*

Start the bridge connection *connectionName*.

stop_connection *connectionName*

Stop the bridge connection *connectionName*.

Problem determination commands

With the problem determination commands you can modify the settings of log_level, max_log_entries, max_trace_entries, and trace_output. You can also take a heap dump, an FFDC snapshot, or a trace buffer dump.

heap_dump *heapPath*

Create a heap dump and write it to *heapPath*. *heapPath* is either a path, or a filename, relative to the working directory.

log_level config|detail|info|audit|warning|error|severe

log_level is the level of log output required. The log levels are listed in order of increasing importance.

Log messages are written to `stdout` and to the `$SYS/broker/log` topic.

max_log_entries *maxLogEntries*

maxLogEntries is the maximum number of log entries remembered for retrieval by the **trace_dump** command or in an FFDC.

max_trace_entries *maxTraceEntries*

maxTraceEntries is the maximum number of trace entries remembered for retrieval by the **trace_dump** command or in an FFDC.

take_ffdc *tag*

Take a First Failure Data Capture (FFDC) snapshot of the state of the daemon. The snapshot is written to a `.fdc` file in the folder defined by the daemon configuration parameter, `ffdc_output`; see `ffdc_output`. *tag* is embedded in the file for identification purposes.

trace_dump *dumpPath*

Dump the trace buffer to *dumpPath*. *dumpPath* is either a path, or a filename, relative to the working directory.

trace_level minimum|medium|maximum

trace_level is the level of trace taken and stored in an internal buffer.

trace_output off|stdout|stderr|protocol|*tracePath*

trace_output is the destination to write trace entries as they occur. It also controls whether a full trace or just a message trace is taken.

Tracing continues indefinitely until explicitly turned off and results in large files.

The **protocol** setting writes an entry for every MQTT message sent to or received from a client to stdout.

The stdout, stderr and *tracePath* settings write a complete trace to the specified destination.

tracePath is either a path, or a file name, relative to the working directory.

MQXR properties

MQXR property settings are stored in a platform-specific properties file: `mqxr_win.properties` or `mqxr_unix.properties`. You normally configure these properties by using MQSC admin commands or MQ Explorer.

When you start a queue manager for the first time, the template version of the MQXR properties file for your platform is copied from the *mqinstall*`/mqxr/config` directory to the *mqinstall*`/qmgrs/`*qmgr_name*`/mqxr/config` directory.

You do not normally need to edit the MQXR properties file directly, because all properties except one can be configured through MQSC admin commands or MQ Explorer. If you do decide to edit the file directly, stop the queue manager before you make your changes.

The property that you can only set by editing the file directly is **webcontentpath**. If your telemetry client app is a web app, you also need to serve the web app executable JavaScript to the browser. This requirement is explained in The MQTT messaging client for JavaScript and web apps. You use the **webcontentpath** property to specify the directory from which the web app executable files are served:

- By default, **webcontentpath** is not present in the MQXR properties file. If **webcontentpath** is not present, the MQ telemetry server serves the web app executable files from the following default location: *mqinstall*/qmgrs/*qmgr_name*/mqxr/WebContent/*your_client_app*
- if **webcontentpath** specifies a path, the MQ telemetry server serves the web app executable files from that location.
- if **webcontentpath** is present and blank, the MQ telemetry server does not serve the web app executable files.

Related information:

Telemetry (MQXR) service

Security reference

Use the reference information in this section to help you configure security for IBM WebSphere MQ.

Related concepts:

"The API exit" on page 2824
An *API exit* is a program module that monitors or modifies the function of MQI calls. An API exit comprises multiple *API exit functions*, each with its own entry point in the module.

"The API-crossing exit" on page 2825
An *API-crossing exit* is a program that monitors or modifies the function of MQI calls issued by CICS applications on z/OS.

"Certificate validation and trust policy design on UNIX, Linux and Windows systems" on page 2826
WebSphere MQ validates SSL or TLS certificates according to two types of policy, basic, and standard. Standard policy checking conforms to RFC 5280.

"Cryptographic hardware" on page 2840
On UNIX, Linux and Windows systems, WebSphere MQ provides support for a variety of cryptographic hardware using the PKCS #11 interface. On IBM i and z/OS, the operating system provides the cryptographic hardware support.

"WebSphere MQ rules for SSLPEER values" on page 2841
The SSLPEER attribute is used to check the Distinguished Name (DN) of the certificate from the peer queue manager or client at the other end of a IBM WebSphere MQ channel. IBM WebSphere MQ uses certain rules when comparing these values

"Migrating with AltGSKit from WebSphere MQ V7.0.1 to WebSphere MQ V7.1" on page 2842
Perform this task only if you are migrating from WebSphere MQ V7.0.1 using the AltGSKit configuration setting to load an alternative GSKit. The alternative GSKit used by WebSphere MQ V7.0.1 with the AltGSKit setting is separate from the GSKit used by WebSphere MQ V7.1; changes to each GSKit do not affect the other. This is because WebSphere MQ V7.1 uses a private local copy of GSKit in its installation directory and does not support the use of an alternative GSKit.

"CipherSpec mismatches" on page 2845
Both ends of a WebSphere MQ SSL channel must use the same CipherSpec. Mismatches can be detected during the SSL handshake or during channel startup.

"Authentication failures" on page 2845
There are a number common reasons for authentication failures during the SSL handshake.

Related reference:

"GSKit: Digital certificate signature algorithms compliant with FIPS 140-2" on page 2842
The list of digital certificate signature algorithms in GSKit that are compliant with FIPS 140-2

The API exit

An *API exit* is a program module that monitors or modifies the function of MQI calls. An API exit comprises multiple *API exit functions*, each with its own entry point in the module.

Note: The information in this section does not apply to WebSphere MQ for z/OS.

There are two categories of exit function:

An exit function that is associated with an MQI call

There are two exit functions in this category for each MQI call and an additional one for an MQGET call with the MQGMO_CONVERT option. The MQCONN and MQCONNX calls share the same exit functions.

For each MQI call, one of the two exit functions is invoked before the queue manager starts to process the call and the other is invoked after the queue manager has completed processing the call. The exit function for an MQGET call with the MQGMO_CONVERT option is invoked during the MQGET call, after the message has been retrieved from the queue by the queue manager but before any data conversion takes place. This allows, for example, a message to be decrypted before data conversion.

An exit function can inspect and modify any of the parameters on an MQI call. On an MQPUT call, for example, an exit function that is invoked before the processing of the call has started can:

- Inspect and modify the contents of the application data in the message being put
- Change the length of the application data in the message
- Modify the contents of the fields in the message descriptor structure, MQMD
- Modify the contents of the fields in the put message options structure, MQPMO

An exit function that is invoked before the processing of an MQI call has started can suppress the call completely. The exit function for an MQGET call with the MQGMO_CONVERT option can suppress data conversion of the message being retrieved.

Initialization and termination exit functions

There are two exit functions in this category, the initialization exit function and the termination exit function.

The initialization exit function is invoked by the queue manager when an application connects to the queue manager. Its primary purpose is to register exit functions and their entry points with the queue manager and perform any initialization processing. You do not have to register all the exit functions, only those that are required for this connection. When the application disconnects from the queue manager, the registrations are removed automatically.

The initialization exit function can also be used to acquire any storage required by the exit and examine the values of any environment variables.

The termination exit function is invoked by the queue manager when an application disconnects from the queue manager. Its purpose is to release any storage used by the exit and perform any required cleanup operations.

An API exit can issue calls to the MQI but, if it does, the API exit is not invoked recursively a second time. The following exit functions, however, are not able to issue MQI calls because the correct environment is not present at the time the exit functions are invoked:

- The initialization exit function
- The exit function for an MQCONN and MQCONNX call that is invoked *before* the queue manager starts to process the call
- The exit function for the MQDISC call that is invoked *after* the queue manager has completed processing the call
- The termination exit function

An API exit can also use other APIs that might be available; for example, it can issue calls to DB2.

An API exit can be used with a WebSphere MQ client application, but it is important to note that the exit is invoked at the *server* end of an MQI channel. For more information, see Comparing link level security and application level security.

An API exit is written using the C programming language.

To enable an API exit, you must configure it. On IBM i, Windows, UNIX and Linux systems, you do this by editing the WebSphere MQ configuration file, mqs.ini, and the queue manager configuration file, qm.ini, for each queue manager.

For a client, modify the `ApiExitLocal` stanza in the mqclient.ini file to identify API exit routines for a queue manager.

You configure an API exit by providing the following information:

- The descriptive name of the API exit.
- The name of the module and its location; for example, the full path name.
- The name of the entry point for the initialization exit function.
- The sequence in which the API exit is invoked relative to other API exits. You can configure more than one API exit for a queue manager.
- Optionally, any data to be passed to the API exit.

For more information about how to configure an API exit, see Configuring API exits.

For information about how to write an API exit, see Using and writing API exits.

The API-crossing exit

An *API-crossing exit* is a program that monitors or modifies the function of MQI calls issued by CICS applications on z/OS.

Note: The information in this section applies only to CICS applications on z/OS.

The API-crossing exit program is invoked by the CICS adapter and runs in the CICS address space.

The API-crossing exit is invoked for the following MQI calls only:

MQBUFMH
MQCB
MQCB_FUNCTION
MQCLOSE
MQCRTMH
MQCTL
MQDLTMH
MQGET
MQINQ
MQOPEN
MQPUT
MQPUT1
MQSET
MQSTAT

MQSUB
MQSUBRQ

For each MQI call, it is invoked once before the processing of the call has started and once after the processing of the call has been completed.

The exit program can determine the name of an MQI call and can inspect and modify any of the parameters on the call. If it is invoked before an MQI call is processed, it can suppress the call completely.

The exit program can use any of the APIs that a CICS task-related user exit can use; for example, the IMS, DB2, and CICS APIs. It can also use any of the MQI calls except MQCONN, MQCONNX, and MQDISC. However, any MQI calls issued by the exit program do not invoke the exit program a second time.

You can write an API-crossing exit in any programming language supported by WebSphere MQ for z/OS.

Before an API-crossing exit can be used, the exit program load module must be available when the CICS adapter connects to a queue manager. The load module is a CICS program that must be named CSQCAPX and reside in a library in the DFHRPL concatenation sequence. CSQCAPX must be defined in the CICS system definition file (CSD), and the program must be enabled.

An API-crossing exit can be managed using the CICS adapter control panels, CKQC. When CSQCAPX is loaded, a confirmation message is written to the adapter control panels or to the system console. The adapter control panels can also be used to enable or disable the exit program.

For more information about how to write and implement an API-crossing exit, see 'The CICS-WebSphere MQ Adapter' section in the CICS Transaction Server for z/OS Version 4.1 product documentation at: CICS Transaction Server for z/OS Version 4.1, The CICS-WebSphere MQ adapter.

Certificate validation and trust policy design on UNIX, Linux and Windows systems

WebSphere MQ validates SSL or TLS certificates according to two types of policy, basic, and standard. Standard policy checking conforms to RFC 5280.

The information in these topics applies to the following systems:

- WebSphere MQ for UNIX and Linux systems
- WebSphere MQ for Windows systems

The following terms are used in this section:

Certificate policy
Determines which fields in a certificate are understood and processed.

OCSP policy
Determines which fields in an OCSP request or response are understood and processed.

CRL policy
Determines which fields in a certificate revocation list are understood and processed.

Path validation policy
Determines how the certificate, OCSP, and CRL policy types interact with each other to determine whether a certificate chain (a trust point "RootCA" to an end-entry "EE") is valid.

The basic and standard path validation policies are described separately because it reflects the implementation within WebSphere MQ for UNIX, Linux and Windows systems. However, the standard

OCSP and CRL policies are the same as the basic policies, and the standard certificate policy is an extended version of the basic policy, so these policies are not described separately.

By default, WebSphere MQ applies basic policy validation first. If basic policy validation fails, WebSphere MQ applies standard policy (RFC 5280) validation. If basic policy validation succeeds, standard policy validation is not applied. Thus, a validation failure means that both basic and standard policy validation failed, possibly for different reasons. A validation success means that either basic policy validation succeeded and standard policy validation was therefore not applied, or basic policy validation failed and standard policy validation succeeded.

Enforcing strict RFC 5280 compliance

To enforce strict RFC 5280 compliance, use the certificate validation policy configuration setting. This setting allows you to disable the basic policy, so that only the standard RFC 5280 policy is used. For more information about the certificate validation policy configuration setting, see Certificate validation policies in WebSphere MQ.

The following examples are digital certificates which are accepted by the basic certificate validation policy, but which are rejected by the RFC 5280 compliant standard policy. In order for a digital certificate chain to be trusted, the entire chain must satisfy the configured validation policy.

To view the full details of a digital certificate, use the **runmqakm** command:

```
runmqakm -cert -details -db key.kdb -pw password -label certificate_label
```

A certificate which has trust status enabled in the **runmqakm** output is not necessarily trusted for use in an SSL or TLS handshake. Trust status enabled means that the certificate is eligible to be used as a CA certificate to verify other certificates, if the certificate also satisfies the rules of the certificate validation policy. For more information about the RFC 5280 compliant standard certificate validation policy, see "Standard path validation policy" on page 2836.

Example certificate 1 - incorrect key usage

This example shows a certificate where the key usage field does not comply with the standard certificate validation policy rules for a CA certificate. One of the requirements for a certificate to be valid for use as a CA certificate is that the key usage field must indicate that it is permitted to sign other certificates using the keyCertSign flag. A certificate without this flag cannot be used as a CA certificate.

```
Label : root
Key Size : 1024
Version : X509 V3
Serial : 54cb6f740c7ee410
Issuer : CN=Example Root CA,O=Example,C=GB
Subject : CN=Example Root CA,O=Example,C=GB
Not Before : 9 February 2012 17:19:00 GMT
Not After : 1 October 2019 18:19:00 GMT+01:00
Public Key
    30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
    05 00 03 81 8D 00 30 81 89 02 81 81 00 CC 44 D9
    25 6D 26 1C 9D B9 FF DE B8 AC 44 AB E3 64 80 44
    AF BE E0 00 93 53 92 33 F8 7E BD D7 71 ED 21 52
    24 75 DF D6 EE 3C 54 97 84 29 EA 93 4C 4A D1 19
    5D C1 A0 82 F5 74 E1 AD D9 87 10 D5 6A 2B 6F 90
    04 0F 7E 6E 85 6D 32 99 33 9C D9 BB 57 86 DE 68
    23 C9 F2 6D 53 E3 F5 FF D1 0B E7 23 19 3A F6 70
    6B C8 C7 EB DB 78 8E 8C 9E 55 58 66 B6 31 DB 40
    5F 6A 97 AB 12 D7 E2 3E 2E 79 EE 78 7B 02 03 01
    00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    EE 68 D4 4F 73 4F F4 21 DE 1A 01 11 5E DE B1 B8
    DF 40 AA D8
```

```
Fingerprint : MD5 :
    50 B5 E9 B2 D7 35 05 6A DC 6D 4B 1E B2 F2 DF A4
Fingerprint : SHA256 :
    B4 D7 6E C4 47 26 24 C7 4F 41 C3 83 03 6F 5C C7
    07 11 61 E0 0E 36 59 1F 1C E6 69 39 2D 18 05 D2
Extensions
    basicConstraints
        ca = true
        pathLen = 1239876
        critical
    key usage: encipherOnly
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
Value
    9D AE 54 A9 9D 68 01 68 15 B5 53 9F 96 C9 5B D1
    52 40 DB CB 33 AF FD B9 26 D5 90 3F 1E 0B FC A6
    D9 8C 04 90 EB AA FD A8 7A 3C AB 60 5F 20 4F 0D
    7B 73 41 27 6A 2B BF 8C 99 91 B6 49 96 82 6A 24
    0A E8 B9 A5 AF 69 3D 2C A3 3C C8 12 39 FB 56 58
    4E 2A FE AC AC 10 89 53 B1 8F 0F C0 50 BF 5E 00
    91 64 B4 A1 4C 9A 4E D5 1F 38 7C AD 32 A9 8A E1
    91 16 2C 6D 1E 4A CA 99 8D CC 22 CD BF 90 49 FC
Trust Status : Enabled
```

In this example, the key usage field contains only the encipherOnly flag. The keyCertSign flag is not set, so this certificate is not permitted to sign other certificates. It therefore cannot be used as a CA certificate.

Example certificate 2 - missing basic constraints extension

This example shows a certificate which lacks the basic constraints extension. The basic constraints extension is used to indicate whether this certificate is permitted for use as a CA. It is also used to indicate the maximum length of any certificate chain which can be signed by the certificate. The standard certificate validation policy requires that the certificate has a basic constraints extension with the isCA flag set in order to be used as a CA.

```
Label : root
Key Size : 1024
Version : X509 V3
Serial : 1c7dfea316570bf6
Issuer : CN=Second Example Root CA,O=Example,C=GB
Subject : CN=Second Example Root CA,O=Example,C=GB
Not Before : 9 February 2012 17:18:22 GMT
Not After : 1 October 2019 18:18:22 GMT+01:00
Public Key
    30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
    05 00 03 81 8D 00 30 81 89 02 81 81 00 B2 70 49
    7C AE 1B A7 B3 06 49 6C 99 19 BC A8 77 BE 86 33
    21 6B C9 26 CC A6 28 52 9F 7B CF 03 A4 37 A7 4D
    6B 06 AA ED 7D 58 E3 70 F3 F7 C1 06 DA E8 27 C6
    3D 1B AC FA EF AA 59 7A 9A AB C1 14 4E AF 13 14
    4B 71 CA 8D FE C3 F5 2F E8 AC AD EF 21 80 6D 12
    89 4A 2A 84 AA 9D E0 4F C1 93 B1 3E 16 E8 3C 75
    39 2A 74 1E 90 CC B1 C3 2B 1D 55 26 76 D2 65 C1
    06 47 2A BF 79 96 42 76 A9 6E 65 88 5F 02 03 01
    00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    33 9F A1 81 43 F1 43 95 48 A5 66 B4 CD 98 E8 15
    9C B3 CA 90
Fingerprint : MD5 :
    91 EA D9 C0 2C 05 5B E2 CD 0B F6 DD 8A 11 44 23
Fingerprint : SHA256 :
    62 46 35 0B 0E A1 A7 2A D5 74 70 0F AA 47 9A 9C
    6B 80 1B F1 0B 4C 81 05 85 0E 91 11 A4 21 D2 34
Extensions
    key usage: digitalSignature, keyCertSign
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
```

```
Value
    79 34 BA 5B 6F DC 06 A3 99 24 4E 8A 2B 27 05 47
    0D 4D BE 6A 77 D1 1D 5F 54 82 9D CC F6 92 D4 9A
    AB 4D B6 DD 6E AD 86 C3 6A A3 32 E3 B3 ED E0 62
    4A EB 51 08 AC BE 49 9E 9C D7 FE AE C8 9D 17 16
    68 31 6B F4 BA 74 1E 4F 5F 05 48 9F E7 46 BA DC
    17 7A 60 88 F8 5B DB 3C 51 D4 98 97 28 82 CF 36
    47 DA D2 0F 47 FF 70 EA 45 3A 49 66 E6 E2 F9 67
    2C C8 3E 24 A2 3B EC 76 1F D6 31 2B BD A9 B5 08
Trust Status : Enabled
```

In this example, the certificate lacks the basic constraints field entirely. Therefore this certificate cannot be used as a CA certificate.

Example certificate 3 - intermediate CA with old version of X.509

This example shows an intermediate CA certificate which is at X.509 version 1. The standard certificate validation policy requires that all intermediate CA certificates must be at least X.509 version 3. Root CA certificates are exempt from this requirement as there are still some commonly used version 1 root CA certificates in existence. However, this exemption might change in future.

```
Label : intermediate
Key Size : 1024
Version : X509 V1
Serial : 02
Issuer : CN=Test Root CA,O=Example,C=GB
Subject : CN=Test Intermediate CA,O=Example,C=GB
Not Before : 10 February 2012 17:33:45 GMT
Not After : 11 April 2018 18:33:45 GMT+01:00
Public Key
    30 81 9F 30 0D 06 09 2A 86 48 86 F7 0D 01 01 01
    05 00 03 81 8D 00 30 81 89 02 81 81 00 C0 07 C2
    D0 9F 84 DB 7C 20 8F 51 F9 C2 1A 3F CF E2 D7 F2
    F1 56 F2 A4 8F 8F 06 B7 3B 01 31 DE 7C CC 03 63
    AA D3 2F 1C 50 15 E3 56 80 40 7D FF 75 87 D3 F3
    00 89 9A 26 F5 57 05 FA 4F ED 3B DD 93 FA F2 DF
    38 26 D4 3A 92 51 CC F3 70 27 42 7A 9F AD 51 45
    67 B7 AE 11 AD 4F 2D AB D2 CF 73 E6 F0 45 92 F0
    47 16 66 7E 01 C7 76 A3 7B EC D2 76 3F E5 15 EC
    D7 72 2C FE 14 F5 78 83 AA C4 20 AB F7 02 03 01
    00 01
Public Key Type : RSA (1.2.840.113549.1.1.1)
Fingerprint : SHA1 :
    DE BB 75 4B 14 E1 44 B9 B6 44 33 97 49 D0 82 6D
    81 F2 2F DE
Fingerprint : MD5 :
    72 49 44 42 E2 E6 89 F1 CC 37 C9 F6 B5 8F F3 AE
Fingerprint : SHA256 :
    83 A4 52 AF 49 34 F1 DC 49 E6 95 AE 93 67 80 13
    C2 64 D9 26 22 A0 E8 0A 5A A9 71 EC E8 33 E1 D1
Signature Algorithm : SHA256WithRSASignature (1.2.840.113549.1.1.11)
Value
    40 4A 09 94 A0 18 07 5E 96 D7 A6 52 6B 8D 20 50
    E8 91 F7 7E EA 76 B4 08 DF 76 66 1F FA FF 91 79
    2E E0 66 8B 9F 40 FA 14 13 79 81 DB 31 A5 55 1D
    44 67 41 F4 EA 1A F7 83 4F 21 F4 43 78 4E F8 5E
    6F B2 B8 3A F7 6B B4 F5 C6 F8 EB 4C BF 62 6F 3E
    C7 20 EC 53 B3 40 51 36 C1 0A 4E 73 ED 74 D1 93
    02 C5 FB 61 F7 87 64 A5 94 06 7D 25 7C E3 73 DD
    08 D4 07 D0 A4 3F 77 88 12 59 DB A4 DB 68 8F C1
Trust Status : Enabled
```

In this example, the version field is X.509 V1. This certificate is an X.509 version 1 certificate and therefore cannot be used as an intermediate CA.

Basic and standard certificate policies

The basic and standard certificate policies support the same fields: the standard policy supports additional certificate extensions.

The supported fields for both the basic and standard policies are as follows:

- OuterSigAlgID[11]
- Signature[12]
- Version
- SerialNumber
- InnerSigAlgID[13]
- Issuer
- Validity
- SubjectName
- SubjectPublicKeyInfo
- IssuerUniqueID
- SubjectUniqueID

The supported extensions for the basic policy are as follows. Where an entry is marked as "not supported", WebSphere MQ does not attempt to process extensions containing a field of that specific type, but does process other types of the same extension.

- AuthorityKeyID
- AuthorityInfoAccess
- SubjectKeyID
- IssuerAltName
- SubjectAltName
- KeyUsage
- BasicConstraints
- PrivateKeyUsage
- CRLDistributionPoints
  - DistributionPoint
    - DistributionPointName (X.500 Name and LDAP Format URI only)
    - NameRelativeToCRLIssuer (not supported)
    - Reasons (ignored)
    - CRLIssuer fields (not supported)

The supported extensions for the standard policy are all those listed for the basic policy and those in the following list. Where an entry is marked as "not supported", WebSphere MQ does not attempt to process extensions containing a field of that specific type, but does process other types of the same extension.

- NameConstraints
- ExtendedKeyUsage
- CertificatePolicies
  - PolicyInformation
    - PolicyIdentifier

---

11. This field is called *signatureAlgorithm* in RFC 5280.
12. This field is called *signatureValue* in RFC 5280.
13. This field is called *signature* in RFC 5280.

    - PolicyQualifiers (not supported)
- PolicyMappings
- PolicyConstraints

Basic and standard OCSP policies

The basic and standard OCSP policies support the same fields.

The supported fields for a request are as follows. Where an entry is marked as "not supported", WebSphere MQ does not attempt to process a request containing a field of that specific type, but does process other requests containing the same higher-level field.

- Signature (Optional)
- Version (Version 1 Only)
- RequesterName (Optional)
- RequestList (single request only)
  - CertID [14]
  - singleRequestExtensions (not supported)
- RequestExtensions
  - Nonce (if enabled)

The supported fields for a response are as follows:

- ResponseStatus
- Response
  - responseType (id-pkix-ocsp-basic)
  - BasicOCSPResponse
    - Signature
    - Certs
      - Extensions
      - extendedKeyUsage
        - id-kp-OCSPSigning
      - id-pkix-ocsp-nocheck
    - ResponseData
      - Version (Version 1 Only)
      - ResponderID (by name or by hash)
      - ProducedAt (ignored)
      - Responses (multiple responses supported)
        - SingleResponse
          - certID
          - certStatus
            - RevokedInfo (ignored)
          - thisUpdate (ignored)
          - nextUpdate
          - singleExtensions (ignored)
      - responseExtensions
        - Nonce (if enabled)

14. This field is called reqCert in RFC 2560

Basic and standard CRL policies

The basic and standard CRL policies support the same fields and extensions.

The supported fields for these policies are as follows:

- OuterSigAlgID[15]
- Signature[16]
- Version
- InnerSigAlgID[17]
- Issuer
- ThisUpdate
- NextUpdate
- RevokedCertificate
  - UserCertificate
  - RevocationDate

There are no supported CRLEntry extensions.

The supported CRL extensions for these policies are as follows. Where an entry is marked as "not supported", WebSphere MQ does not attempt to process extensions containing a field of that specific type, but does process other types of the same extension.

- AuthorityKeyID
- IssuerAltName
- CRLNumber
- IssuingDistributionPoint
  - DistributionPoint
  - DistributionPointName
    - FullName (X.500 Name and LDAP Format URI only)
    - NameRelativeToCRLIssuer (not supported)
  - Reasons (ignored)
  - CRLIssuer
  - OnlyContainsUserCerts (not supported)
  - OnlyContainsCACerts (not supported)
  - OnlySomeReasons (not supported)
  - IndirectCRL[18] (rejected)

15. This field is called *signatureAlgorithm* in RFC 5280.
16. This field is called *signatureValue* in RFC 5280.
17. This field is called *signature* in RFC 5280.
18. IndirectCRL extensions will result in CRL validation failing. IndirectCRL extensions must not be used because they cause identified certificates to not be rejected.

Basic path validation policy

The basic path validation policy determines how the certificate, OCSP, and CRL policy types interact with each other to determine if a certificate chain is valid.

The validation of a chain is performed in the following manner (but not necessarily in the following order):

1. Ensure that the name of the certificate's issuer is equal to the subject name in the previous certificate, and that there is not an empty issuer name in this certificate or the previous certificate subject name. If no previous certificate exists in the path and this is the first certificate in the chain, ensure that the issuer and subject name are identical and that the trust status is set for the certificate[19].

   Note: WebSphere MQ for UNIX, Linux and Windows systems will fail path validation in situations where the previous certificate in a path has the same subject name as the current certificate.
2. Ensure that the signature algorithm used to actually sign the certificate matches the signature algorithm indicated within the certificate, by ensuring that the issuer signature algorithm identifier in the certificate matches the algorithm identifier in the signature data.
3. Ensure that the certificate was signed by the issuer, using the subject public key from the previous certificate in the path to verify the signature on the certificate. If no previous certificate exists and this is the first certificate, use the subject public key of the certificate to verify the signature on it. WebSphere MQ supports DSA and RSA signature algorithms; however it does not support DSA Parameter Inheritance.
4. Ensure that the certificate is a known X509 version, unique IDs are not present for version 1 certificates, and extensions are not present for version 1 and version 2 certificates.
5. Ensure that the certificate has not expired, or not been activated yet, and that its validity period is good[20].
6. Ensure that there are no unknown critical extensions or any duplicate extensions.
7. Ensure that the certificate has not been revoked. Here, the following operations apply:
   a. If the OCSP connection is enabled and a Responder Address is configured or the Certificate has a valid AuthorityInfoAccess extension specifying a HTTP format GENERALNAME_uniformResourceID check revocation status with OCSP.
   b. If revocation status from 7a above is undetermined the CRLDistributionPoints extension is checked for a list of X.500 distinguished name GENERALNAME_directoryname and URI GENERALNAME_uniformResourceID. Only LDAP, HTTP and FILE format URIs are supported. If the extension is not present, or use of the CRLDistributionPoints extension results in undetermined status and the extension is not Critical, the certificate's issuer's name is used to query revocation status. A CRL database (LDAP) is then queried for CRLs. If the certificate is not the last certificate, or if the last certificate has the basic constraint extension with the "isCA" flag turned on, the database is queried for ARLs and CRLs instead. If CRL checking is enabled, and no CRL database can be queried, the certificate is treated as revoked. Currently, the X500 directory name form and the LDAP/HTTP/FILE URI forms are the only supported name forms used to look up CRLs and ARLs[21].

      Note: RelativeDistinguishedNames are not supported.

---

19. Trust status is an administrative setting in the key database file. You can access and alter the trust status of a particular signer certificate in iKeyman. Select the required certificate from the signer list and click **View/Edit...**. The **Set the certificate as a trusted root** check box on the resulting panel indicates the trust status. You can also set Trust status using iKeycmd with the `-trust` flag on the **-cert -modify** command. For further information about this command, see Managing keys and certificates.
20. There are no checks to ensure the subject's validity is within bounds of the issuer's validity. This is not required, and it has been shown that certificates from some CAs do not pass such a check.
21. After they are retrieved from the database, ARLs are evaluated in exactly the same fashion as CRLs. Many CAs do not issue ARLs. However, WebSphere MQ will look for ARLs and CRLs if checking a CA certificate for revocation status.

c. If revocation status from both 7a on page 2833 and 7b on page 2833 is undetermined, WebSphere MQ checks the *OCSPAuthentication* configuration setting to decide whether to allow the connection.[22]

8. If the issuerAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:
   - rfc822
   - DNS
   - directory
   - URI
   - IPAddress(v4/v6)
9. If the subjectAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:
   - rfc822
   - DNS
   - directory
   - URI
   - IPAddress(v4/v6)
10. If the KeyUsage extension is critical on a non-EE certificate, ensure that the keyCertSign flag is on, and ensure that if the BasicConstraints extension is present, the "isCA" flag is true.
11. If the BasicConstraints extension is present, the following checks are made:
    - If the "isCA" flag is false, ensure the certificate is the last certificate in the chain and that the pathLength field is not present.
    - If the "isCA" flag is true and the certificate is NOT the last certificate in the chain, ensure that the number of certificates until the last certificate in the chain is not greater than the pathLength field.
12. The AuthorityKeyID extension is not used for path validation, but is used when building the certificate chain.
13. The SubjectKeyID extension is not used for path validation, but is used when building the certificate chain.
14. The PrivateKeyUsagePeriod extension is ignored by the validation engine, because it cannot determine when the CA actually signed the certificate. The extension is always non-critical and therefore can be safely ignored.

An OCSP Response is also validated to ensure that the response itself is valid. Validation is performed in the following manner (but not necessarily the following order):

1. Ensure that response status is `Successful` and the response type is PKIX_AD_OCSP_basic.r
2. Ensure that response version data is present and the response is the correct version (Version 1)
3. Ensure that the response is correctly signed. The signature will be rejected if the signer does not meet at least one of the following criteria:
   - The signer matches a local configuration of OCSP signing authority[23] for the certificate.
   - The signer is using the CA key for which the public key is contained in the CA certificate, that is, the CA itself is directly signing the response.
   - The signer is a direct sub-ordinate of the CA that signed the certificate for which revocation information is being checked and is authorized by the CA by including the value of id-ad-ocspSigning in an ExtendedKeyUsage extension.

---

22. If *OCSPAuthentication* is set to `WARN`, WebSphere MQ logs the unknown revocation status and allows the connection to continue.

23. This is a Certificate in the KeyStore a user has installed and that has Trust Status set.

Note: Revocation checking of the response signer certificate is not performed if the id-pkix-ocsp-nocheck extension is present.

4. Ensure that response hash algorithm, serialNumber, issuerNameHash, and issuerKeyHash match those of the request.
5. Ensure that the response has not expired, that is, that the nextUpdate time is greater than the current time.[24]
6. Ensure that the certificate has valid revocation status.

The validation of a CRL is also performed to ensure that the CRL itself is valid, and is performed in the following manner (but not necessarily the following order):

1. Ensure that the signature algorithm used to actually sign the CRL matches the signature algorithm indicated within the CRL, by ensuring that the issuer signature algorithm identifier in the CRL matches the algorithm identifier in the signature data.
2. Ensure that the CRL was signed by the issuer of certificate in question, verifying that the CRL has been signed with the key of the certificate issuer.
3. Ensure that the CRL has not expired[25], or not been activated yet, and that its validity period is good.
4. Ensure that if the version field is present, it is version 2. Otherwise the CRL is version 1 and must not have any extensions. However, WebSphere MQ for UNIX, Linux and Windows systems only verifies that no critical extensions are present for a version 1 CRL.
5. Ensure that the certificate in question is on the revokedCertificates field list and that the revocation date is not in the future.
6. Ensure that there are no duplicate extensions.
7. If unknown critical extensions, including critical entry extensions, are detected in the CRL, this causes identified certificates to be treated as revoked[26] (provided the CRL passes all other checks).
8. If the authorityKeyID extension in the CRL and the subjectKeyID in the CA certificate are present and if the keyIdentifier field is present within the authorityKeyID of the CRL, match it with the CACertificate's subjectKeyID.
9. If the issuerAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:
   - rfc822
   - DNS
   - directory
   - URI
   - IPAddress(v4/v6)

---

24. If no current OCSP responses are returned from the responder, WebSphere MQ will attempt to use out of date responses in determining the revocation status of a Certificate. WebSphere MQ attempts to use out of date Responses so that security will not be adversely reduced.

25. If no current CRLs are found, WebSphere MQ for UNIX, Linux and Windows systems will attempt to use out of date CRLs to determine the revocation status of a Certificate. It is not clearly specified in RFC 5280 what action to take in the event of no current CRLs. WebSphere MQ for UNIX, Linux and Windows systems attempt to use out of date CRLs so that security will not be adversely reduced.

26. ITU X.509 and RFC 5280 are in conflict in this case because the RFC mandates that CRLs with unknown critical extensions must fail validation. However, ITU X.509 requires that identified certificates must still be treated as revoked provided the CRL passes all other checks. WebSphere MQ for UNIX, Linux and Windows systems adopt the ITU X.509 guidance so that security will not be adversely reduced.

    A potential scenario exists where the CA that issues a CRL might set an unknown critical extension to indicate that even though all other validation checks are successful, a certificate which is identified must not be considered revoked and thus not rejected by the application. In this scenario, following X.509, WebSphere MQ for UNIX, Linux and Windows systems will function in a fail-secure mode of operation. That is, they might reject certificates that the CA did not intend to be rejected and therefore might deny service to some valid users. A fail-insecure mode ignores a CRL because it has an unknown critical extension and therefore certificates that the CA intended to be revoked are still accepted. The administrator of the system should then query this behavior with the issuing CA.

10. If the issuingDistributionPoint extension is present in the CRL, process as follows:
    - If the issuingDistributionPoint specifies an InDirectCRL then fail the CRL validation.
    - If the issuingDistributionPoint indicates that a CRLDistributionPoint is present but no DistributionPointName is found, fail the CRL validation
    - If the issuingDistributionPoint indicates that a CRLDistributionPoint is present and specifies a DistributionPointName ensure that it is a GeneralName or LDAP format URI that matches the name given by the certificate's CRLDistributionPoint or the certificate's issuer's name. If the DistributionPointName is not a GeneralName then the CRL validation will fail.

      Note: RelativeDistinguishedNames are not supported and will fail CRL validation if encountered.

Standard path validation policy

The standard path validation policy determines how the certificate, OCSP, and CRL policy types interact with each other to determine if a certificate chain is valid. Standard policy checking conforms to RFC 5280.

Path validation uses the following concepts:

- A certification path of length *n*, where the trust point or root certificate is certificate 1, and the EE is *n*.
- A set of initial policy identifiers (each comprising a sequence of policy element identifiers), that identifies one or more certificate policies, any one of which is acceptable for the purposes of certification path processing, or the special value "any-policy". Currently this is always set to "any-policy".

  Note: WebSphere MQ for UNIX, Linux and Windows systems only supports policy identifiers that are created by WebSphere MQ for UNIX, Linux and Windows systems.
- Acceptable policy set: a set of certificate policy identifiers comprising the policy or policies recognized by the public key user, together with policies deemed equivalent through policy mapping. The initial value of the acceptable policy set is the special value "any-policy".
- Constrained subtrees: a set of root names defining a set of subtrees within which all subject names in subsequent certificates in the certification path can fall. The initial value is "unbounded".
- Excluded subtrees: a set of root names defining a set of subtrees within which no subject name in subsequent certificates in the certification path can fall. The initial value is "empty".
- Explicit policy: an integer which indicates if an explicit policy identifier is required. The integer indicates the first certificate in the path where this requirement is imposed. When set, this variable can be decreased, but cannot be increased. (That is, if a certificate in the path requires explicit policy identifiers, a later certificate cannot remove this requirement.) The initial value is n+1.
- Policy mapping: an integer which indicates if policy mapping is permitted. The integer indicates the last certificate on which policy mapping may be applied. When set, this variable can be decreased, but cannot be increased. (That is, if a certificate in the path specifies policy mapping is not permitted, it cannot be overridden by a later certificate.) The initial value is n+1.

The validation of a chain is performed in the following manner (but not necessarily the following order):

1. The information in the following paragraph is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

   Ensure that the name of the certificate's issuer is equal to the subject name in the previous certificate, and that there is not an empty issuer name in this certificate or the previous certificate subject name. If no previous certificate exists in the path and this is the first certificate in the chain, ensure that the issuer and subject name are identical and that the trust status is set for the certificate[27].

27. Trust status is an administrative setting in the key database file. You can access and alter the trust status of a particular signer certificate in iKeyman. Select the required certificate from the signer list and click **View/Edit...**. The **Set the certificate as a trusted root** check box on the resulting panel indicates the trust status. You can also set Trust status using iKeycmd with the `-trust` flag on the **-cert -modify** command. For further information about this command, see Managing keys and certificates.

If the certificate does not have a subject name, the subjectAltName extension must be present and critical.

2. The information in the following paragraph is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

   Ensure that the signature algorithm used to actually sign the certificate matches the signature algorithm indicated within the certificate, by ensuring that the issuer signature algorithm identifier in the certificate matches the algorithm identifier in the signature data.

   If both the certificate's issuersUniqueID and the issuer's subjectUniqueID are present, ensure they match.

3. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

   Ensure that the certificate was signed by the issuer, using the subject public key from the previous certificate in the path to verify the signature on the certificate. If no previous certificate exists and this is the first certificate, use the subject public key of the certificate to verify the signature on it.

4. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

   Ensure that the certificate is a known X509 version, unique IDs are not present for version 1 certificates and extensions are not present for version 1 and version 2 certificates.

5. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

   Ensure that the certificate has not expired, or not been activated yet, and that its validity period is good[28]

6. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

   Ensure that there are no unknown critical extensions, nor any duplicate extensions.

7. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

   Ensure that the certificate has not been revoked. Here, the following operations apply:

   a. If the OCSP connection is enabled and a Responder Address is configured or the Certificate has a valid AuthorityInfoAccess extension specifying an HTTP format GENERALNAME_uniformResourceID check revocation status with OCSP.

      1) WebSphere MQ for UNIX and Windows systems allows the OCSP Request to be optionally signed for preconfigured responders but this has otherwise no impact on OCSP Response processing.

   b. If revocation status from 7a is undetermined the CRLDistributionPoints extension is checked for a list of X.500 distinguished name GENERALNAME_directoryname and URI GENERALNAME_uniformResourceID. If the extension is not present, the certificate's issuer's name is used. A CRL database (LDAP) is then queried for CRLs. If the certificate is not the last certificate, or if the last certificate has the basic constraint extension with the "isCA" flag turned on, the database is queried for ARL's and CRL's instead. If CRL checking is enabled, and no CRL database can be queried, the certificate is treated as revoked. Currently, the X500 directory name form and the LDAP/HTTP/FILE URI forms are the only supported name forms used to look up CRLs and ARLs15.

28. There are no checks to ensure the subject's validity is within bounds of the issuer's validity. This is not required, and certificates from some CAs have been shown to not pass such a check.

Note: RelativeDistinguishedNames are not supported.

8.

The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

If the subjectAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:

- rfc822
- DNS
- directory
- URI
- IPAddress(v4/v6)

9. Ensure that the subject name and subjectAltName extension (critical or noncritical) is consistent with the constrained and excluded subtrees state variables.
10. If the EmailAddress OID is present in the subject name field as an IA5 string, and there is no subjectAltName extension, the EmailAddress must be consistent with the constrained and excluded subtrees state variable.
11. Ensure that policy information is consistent with the initial policy set :
    a. If the explicit policy state variable is less than or equal to the current certificate's numeric sequence value, a policy identifier in the certificate shall be in the initial policy set.
    b. If the policy mapping variable is less than or equal to the current certificate's numeric sequence value, the policy identifier cannot be mapped.
12. Ensure that policy information is consistent with the acceptable policy set:
    a. If the certificate policies extension is marked critical[29], the intersection of the policies extension and the acceptable policy set is non-null.
    b. The acceptable policy set is assigned the resulting intersection as its new value.
13. Ensure that the intersection of the acceptable policy set and the initial policy set is non-null. If the special Policy of anyPolicy is present then allow it only if it has not been inhibited by the inhibitAnyPolicy extension at this chain position.
14. If an inhibitAnyPolicy extension is present ensure that it is marked Critical and, if so, set the inhibitAnyPolicy state and chain position to the value of the integer value of the extension provided it is not greater than the current value. This is the number of certificates to allow with an anyPolicy Policy before disallowing the anyPolicy Policy.
15. The following steps are performed for all certificates except the last one:
    a. If the issuerAltName extension is marked critical, ensure that the name forms are recognized. The following general name forms are currently recognized:
       - rfc822
       - DNS
       - directory
       - URI
       - IPAddress(v4/v6)
    b.
       1) If the BasicConstraints extension is not present, the certificate is only valid as an EE certificate.
       2) If the BasicConstraints extension is present, ensure that the "isCA" flag is true. Note that "isCA" is always checked to ensure it is true to be as part of the chain building itself,

29. This is maintained as a legacy requirement from RFC2459 (6.1 (e)(1))

however this specific test is still made. If the pathLength field is present, ensure the number of certificates until the last certificate is not greater than the pathLength field.

c. If the KeyUsage extension is critical, ensure that the keyCertSign flag is on, and ensure that if the BasicConstraints extension is present, that the "isCA" flag is true[30].

d. If a policy constraints extension is included in the certificate, modify the explicit policy and policy mapping state variables as follows:
   - i. If requireExplicitPolicy is present and has value *r*, the explicit policy state variable is set to the minimum of its current value and the sum of *r* and *i* (the current certificate in the sequence).
   - ii. If inhibitPolicyMapping is present and has value *q*, the policy mapping state variable is set to the minimum of its current value and the sum of *q* and *i* (the current certificate in the sequence).

e. If the policyMappings extension is present (see 12(b)), ensure that it is not critical, and if policy mapping is allowed, these mappings are used to map between this certificate's policies and its signee's policies.

f. If the nameConstraints extension is present, ensure that it is critical, and that the permitted and excluded subtrees adhere to the following rules before updating the chain's subtree's state in accordance with the algorithm described in RFC 5280 section 6.1.4 part (g):
   1) The minimum field is set to zero.
   2) The maximum field is not present.
   3) The base field name forms are recognized. The following general name forms are currently recognized:
      - rfc822
      - DNS
      - directory
      - URI
      - IPAddress(v4/v6)

16. The ExtendedKeyUsage extension is not checked by WebSphere MQ.

17. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

    The AuthorityKeyID extension is not used for path validation, but is used when building the certificate chain.

18. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

    The SubjectKeyID extension is not used for path validation, but is used when building the certificate chain.

19. The following information is consistent with the basic path validation policy described in “Basic path validation policy” on page 2833:

    The PrivateKeyUsagePeriod extension is ignored by the validation engine, because it cannot determine when the CA actually signed the certificate. The extension is always non-critical and therefore can be safely ignored.

30. This check is in fact redundant because of step (b), but the check is still made.

Cryptographic hardware

On UNIX, Linux and Windows systems, WebSphere MQ provides support for a variety of cryptographic hardware using the PKCS #11 interface. On IBM i and z/OS, the operating system provides the cryptographic hardware support.

For a list of currently supported cryptography cards, see Cryptography Card List for WebSphere MQ.

On all platforms, cryptographic hardware is used at the SSL handshaking stage and at secret key reset.

On IBM i, when you use DCM to create or renew certificates, you can choose to store the key directly in the coprocessor or to use the coprocessor master key to encrypt the private key and store it in a special keystore file.

On z/OS, when you use RACF to create certificates, you can choose to store the key using ICSF (Integrated Cryptographic Service Facility) to obtain improved performance and more secure key storage. During the SSL handshake, and secret key negotiations, a crypto express card, (if available) is used to do RSA operations. After the handshake completes and data begins to flow, data is decrypted in the CPACF and the crypto express card is not used.

On UNIX, Linux and Windows systems, WebSphere MQ support is also provided for SSL cryptographic hardware symmetric cipher operations. When using SSL cryptographic hardware symmetric cipher operations, data sent across an SSL or TLS connection is encrypted/decrypted by the cryptographic hardware product.

On the queue manager, this is switched on by setting the SSLCryptoHardware queue manager attribute appropriately (see ALTER QMGR and Change Queue Manager). On the Websphere MQ MQI client, equivalent variables are provided (see SSL stanza of the client configuration file). The default setting is off.

If this attribute is switched on, WebSphere MQ attempts to use symmetric cipher operations whether the cryptographic hardware product supports them for the encryption algorithm specified in the current CipherSpec or not. If the cryptographic hardware product does not provide this support, WebSphere MQ performs the encryption and decryption of data itself, and no error is reported. If the cryptographic hardware product supports symmetric cipher operations for the encryption algorithm specified in the current CipherSpec, this function is activated and the cryptographic hardware product performs the encryption and decryption of the data sent.

In a situation of low processor usage it is often quicker to perform the encryption/decryption in software, rather than copying the data onto the card, encrypting/decrypting it, and copying it back to the SSL protocol software. Hardware symmetric cipher operations become more useful when the processor usage is high.

On z/OS with cryptographic hardware, support is provided for symmetric cipher operations. This means that the user's data is encrypted and decrypted by the hardware if the hardware has this capability for the CipherSpec chosen, and is configured to support data encryption and decryption.

On IBM i, cryptographic hardware is not used for encryption and decryption of the user's data, even if the hardware has the capability of performing such encryption for the encryption algorithm specified in the current CipherSpec.

WebSphere MQ rules for SSLPEER values

The SSLPEER attribute is used to check the Distinguished Name (DN) of the certificate from the peer queue manager or client at the other end of a IBM WebSphere MQ channel. IBM WebSphere MQ uses certain rules when comparing these values

When SSLPEER values are compared with DNs, the rules for specifying and matching attribute values are as follows:

1. You can use either a comma or a semicolon as a separator.
2. Spaces before or after the separator are ignored. For example:

   ```
   CN=John Smith, O=IBM ,OU=Test , C=GB
   ```
3. The values of attribute types `SERIALNUMBER`, `MAIL`, `E`, `UID OR USERID`, `CN`, `T`, `OU`, `DC`, `O`, `STREET`, `L`, `ST`, `SP`, `S`, `PC`, `C`, `UNSTRUCTUREDNAME`, `UNSTRUCTUREDADDRESS`, `DNQ` are text strings that typically include only the following:
   - Uppercase and lowercase alphabetic characters `A` through `Z` and `a` through `z`
   - Numeric characters `0` through `9`
   - The space character
   - Characters `, . ; ' " ( ) / -`

   To avoid conversion problems between different platforms, do not use other characters in an attribute value. The attribute types, for example `CN`, must be in uppercase characters.
4. Strings containing the same alphabetic characters match irrespective of case.
5. Spaces are not allowed between the attribute type and the `=` character.
6. Optionally, you can enclose attribute values in double quotation marks, for example `CN="John Smith"`. The quotation marks are discarded when matching values.
7. Spaces at either end of the string are ignored unless the string is enclosed in double quotation marks.
8. The comma and semicolon attribute separator characters are considered to be part of the string when enclosed in double quotation marks.
9. The names of attribute types, for example `CN` or `OU`, are considered to be part of the string when enclosed in double quotation marks.
10. Any of the attribute types `ST`, `SP`, and `S` can be used for the State or Province name.
11. Any attribute value can have an asterisk (`*`) as a pattern-matching character at the beginning, the end, or in both places. The asterisk character substitutes for any number of characters at the beginning or end of the string to be matched. This character enables your SSLPEER value specification to match a range of Distinguished Names. For example, `OU=IBM*` matches every Organizational Unit beginning with IBM, such as IBM Corporation.

    The asterisk character can also be a valid character in a Distinguished Name. To obtain an exact match with an asterisk at the beginning or end of the string, the backslash escape character (\) must precede the asterisk: `\*`. Asterisks in the middle of the string are considered to be part of the string and do not require the backslash escape character.
12. The DN can contain multiple OU attributes and multiple DC attributes.
13. When multiple OU attributes are specified, all must exist and be in descending hierarchical order. For an example, see DEFINE CHANNEL.
14. A digital certificate Subject DN can additionally contain multiple attributes of the same type other than OU or DC, but only if the SSLPEER value does not filter on the repeated attribute type. For example, consider a certificate with the following Subject DN:

    ```
    CN=First, CN=Second, O=IBM, C=US
    ```

    An SSLPEER value of `O=IBM, C=US` does not filter on CN, so matches this certificate and allows the connection. An SSLPEER value of `CN=First, O=IBM, C=US` fails to match this certificate because the certificate contains multiple CN attributes. You cannot match multiple CN values.

Related information:

Distinguished Names

Channel authentication records

Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID

GSKit: Digital certificate signature algorithms compliant with FIPS 140-2

The list of digital certificate signature algorithms in GSKit that are compliant with FIPS 140-2

- RSA with SHA-1
- RSA with SHA-224
- RSA with SHA-256
- RSA with SHA-384
- RSA with SHA-512
- DSA with SHA-1
- ECDSA with SHA-1
- ECDSA with SHA-224
- ECDSA with SHA-256
- ECDSA with SHA-384
- ECDSA with SHA-512
- Curve P-192
- Curve P-224
- Curve P-256
- Curve P-384
- Curve P-521
- Curve K-163
- Curve K-233
- Curve K-283
- Curve K-409
- Curve K-571
- Curve B-163
- Curve B-233
- Curve B-283
- Curve B-409
- Curve B-571

Related information:

Digital certificates and CipherSpec compatibility in WebSphere MQ

Migrating with AltGSKit from WebSphere MQ V7.0.1 to WebSphere MQ V7.1

Perform this task only if you are migrating from WebSphere MQ V7.0.1 using the AltGSKit configuration setting to load an alternative GSKit. The alternative GSKit used by WebSphere MQ V7.0.1 with the AltGSKit setting is separate from the GSKit used by WebSphere MQ V7.1; changes to each GSKit do not affect the other. This is because WebSphere MQ V7.1 uses a private local copy of GSKit in its installation directory and does not support the use of an alternative GSKit.

Overview of the main migration steps for AltGSKit

When migrating from WebSphere MQ V7.0.1 utilizing AltGSKit to WebSphere MQ V7.1 there are a number of tasks to be performed to enable the new GSKit to operate successfully. The main steps to consider when migrating:

1. Ensure that no applications require the use of the currently installed alternative GSKit before initiating removal.
2. Remove the AltGSKit setting from the SSL stanza of each queue manager and client configuration file.

3. Restart each MQI client application which is using the alternative GSKit to ensure that no client applications have the alternative GSKit loaded.
4. Issue the REFRESH SECURITY TYPE(SSL) on each queue manager which is using the alternative GSKit to ensure that no queue managers have the alternative GSKit loaded.
5. Uninstall the alternative GSKit as per the platform specific instructions outlined in this topic.
6. Install the alternative GSKit as per the platform specific instructions referred to in this topic.

Removing the AltGSKit setting

Before the alternative GSKit can be uninstalled, the AltGSKit setting must be removed from the SSL stanza of each queue manager and client configuration file.

To view the contents and for further information about the queue manager configuration files, see Queue manager configuration files, qm.ini

For information about the the SSL stanza of the client configuration file, see SSL stanza of the client configuration file.

Once the configuration file has been altered:

1. Restart each MQI client application which is using the alternative GSKit to ensure that no client applications have the alternative GSKit loaded.
2. Issue the REFRESH SECURITY TYPE(SSL) on each queue manager which is using the alternative GSKit to ensure that no queue managers have the alternative GSKit loaded.

Uninstalling GSKit

Here we outline the platform specific instructions for uninstalling the alternative GSKit:

- "Uninstalling GSKit V8 on Windows"
- "Uninstalling GSKit V8 on Linux"
- "Uninstalling GSKit V8 on AIX" on page 2844
- "Uninstalling GSKit V8 on HP-UX" on page 2844
- "Uninstalling GSKit V8 on Solaris" on page 2844

Uninstalling GSKit V8 on Windows

You can uninstall GSKit Version 8 interactively using Add or Remove Programs in the Windows Control Panel. You can uninstall GSKit Version 8 silently using the Windows Installer **msiexec** utility or the GSKit installation file. If you want to use an accessible interface to uninstall GSKit Version 8, use either of the silent uninstallation methods.

Procedure

- To uninstall GSKit V8 by using **msiexec**:
  1. Issue the command

     ```
     msiexec /x PackageName
     ```

     PackageName is one of the values `GSKit8 SSL 32-bit`, `GSKit8 Crypt 32-bit`, `GSKit8 SSL 64-bit`, or `GSKit8 Crypt 64-bit`.
  2. Repeat for each package to be uninstalled.

Uninstalling GSKit V8 on Linux

You can uninstall GSKit V8 using the **rpm** command.

Procedure

Uninstall GSKit v8 by using the following command:

```
rpm -ev gskssl32-8.0.X.Y gskcrypt32-8.0.X.Y
```

X.Y represents the version number of GSKit installed.

On 64-bit Linux platforms run the following additional command:

```
rpm -ev gskssl64-8.0.X.Y gskcrypt64-8.0.X.Y
```

Uninstalling GSKit V8 on AIX

You can uninstall GSKit V8 using the **installp** command.

Procedure

Uninstall GSKit V8 by using the following command:

```
installp -u -g -V2 gskcrypt32.ppc.rte gskssl32.ppc.rte gskcrypt64.ppc.rte gskssl64.ppc.rte
```

Uninstalling GSKit V8 on HP-UX

You can uninstall GSKit Version 8 using the **swremove** command.

Procedure

Uninstall GSKit V8 by using the following command:

```
swremove gskcrypt32 gskssl32 gskcrypt64 gskssl64
```

Uninstalling GSKit V8 on Solaris

You can uninstall GSKit V8 using the **pkgrm** command.

Procedure

Uninstall GSKit V8 by using the following command:

```
pkgrm gsk8ssl32 gsk8cry32 gsk8ssl64 gsk8cry64
```

Installing GSKit on WebSphere MQ V 7.1

On WebSphere MQ V7.1 for Windows, GSKit is automatically installed.

To install GSKit on WebSphere MQ V 7.1 on Linux and UNIX platforms, refer to instructions outlined in the following topics:

- IBM WebSphere MQ components for Linux systems
- IBM WebSphere MQ components for HP-UX systems
- IBM WebSphere MQ components for AIX systems
- IBM WebSphere MQ components for Solaris systems

CipherSpec mismatches

Both ends of a WebSphere MQ SSL channel must use the same CipherSpec. Mismatches can be detected during the SSL handshake or during channel startup.

A CipherSpec identifies the combination of the encryption algorithm and hash function. Both ends of a WebSphere MQ SSL channel must use the same CipherSpec, although they can specify that CipherSpec in a different manner. Mismatches can be detected at two stages:

During the SSL handshake
The SSL handshake fails when the CipherSpec specified by the SSL client is unacceptable to the SSL support at the SSL server end of the connection. A CipherSpec failure during the SSL handshake arises when the SSL client proposes a CipherSpec that is not supported by the SSL provision on the SSL server. For example, when an SSL client running on AIX proposes the `DES_SHA_EXPORT1024` CipherSpec to an SSL server running on IBM i.

During channel startup
Channel startup fails when there is a mismatch between the CipherSpec defined for the responding end of the channel and the CipherSpec defined for the calling end of channel. Channel startup also fails when only one end of the channel defines a CipherSpec.

See Specifying CipherSpecs for more information.

Note: If Global Server Certificates are used, a mismatch can be detected during channel startup even if the CipherSpecs specified on both channel definitions match.

Global Server Certificates are a special type of certificate which require that a minimum level of encryption is established on all the communications links with which they are used. If the CipherSpec requested by the WebSphere MQ channel configuration does not meet this requirement, the CipherSpec is renegotiated during the SSL handshake. This is detected as a failure during WebSphere MQ channel startup as the CipherSpec no longer matches the one specified on the channel.

In this case, change the CipherSpec at both sides of the channel to one which meets the requirements of the Global Server Certificate. To establish whether a certificate that has been issued to you is a Global Server Certificate, contact the certificate authority which issued that certificate.

SSL servers do not detect mismatches when an SSL client channel on UNIX, Linux or Windows systems specifies the `DES_SHA_EXPORT1024` CipherSpec, and the corresponding SSL server channel on UNIX, Linux or Windows systems is using the `DES_SHA_EXPORT` CipherSpec. In this case, the channel runs normally.

Authentication failures

There are a number common reasons for authentication failures during the SSL handshake.

These reasons include, but are not limited to, those in the following list:

A certificate has been found in a Certificate Revocation List or Authority Revocation List
You can check certificates against the revocation lists published by the Certificate Authorities.

A Certificate Authority can revoke a certificate that is no longer trusted by publishing it in a Certificate Revocation List (CRL) or Authority Revocation List (ARL). For more information, see Working with revoked certificates.

An OCSP responder has identified a certificate as Revoked or Unknown
You can check certificates using OCSP. An OCSP responder can return a response of Revoked, indicating that a certificate is no longer valid, or Unknown, indicating that it has no revocation data for that certificate. For more information, see Working with revoked certificates.

A certificate has expired or is not yet active
Each digital certificate has a date from which it is valid and a date after which it is no longer valid, so an attempt to authenticate with a certificate that is outside its lifetime fails.

A certificate is corrupted
If the information in a digital certificate is incomplete or damaged, authentication fails.

A certificate is not supported
If the certificate is in a format that is not supported, authentication fails, even if the certificate is still within its lifetime.

The SSL client does not have a certificate
The SSL server always validates the client certificate if one is sent. If the SSL client does not send a certificate, authentication fails if the end of the channel acting as the SSL server is defined:

- With the SSLCAUTH parameter set to REQUIRED or
- With an SSLPEER parameter value

There is no matching CA root certificate or the certificate chain is incomplete
Each digital certificate is issued by a Certificate Authority (CA), which also provides a root certificate that contains the public key for the CA. Root certificates are signed by the issuing CA itself. If the key repository on the computer that is performing the authentication does not contain a valid root certificate for the CA that issued the incoming user certificate, authentication fails.

Authentication often involves a chain of trusted certificates. The digital signature on a user certificate is verified with the public key from the certificate for the issuing CA. If that CA certificate is a root certificate, the verification process is complete. If that CA certificate was issued by an intermediate CA, the digital signature on the intermediate CA certificate must itself be verified. This process continues along a chain of CA certificates until a root certificate is reached. In such cases, all certificates in the chain must be verified correctly. If the key repository on the computer that is performing the authentication does not contain a valid root certificate for the CA that issued the incoming root certificate, authentication fails.

However, certain SSL implementations such as GSKit, DCM, and RACF validate the certificates as long as the trust anchor (ROOT CA) is present, with some of the intermediate CA not present in the trust chain. Therefore, it is important to ensure that the server-side certificate store contains the complete trust chain. Also, the technique of selectively removing signer (CA) certificates must not be used to control connectivity to the queue manager.

For more information, see How certificate chains work.

For more information about the terms used in this topic, see:

- Secure Sockets Layer (SSL) and Transport Layer Security (TLS) concepts
- Digital certificates

Monitoring reference

Use the reference information in this section to help you monitor IBM WebSphere MQ.

- "Structure data types" on page 2847
- "Object attributes for event data" on page 2872

Related information:
Monitoring and performance

Structure data types

Use this topic to understand the structure data types used in the message data that WebSphere MQ monitoring techniques generate.

The following topics describe in a language-independent form the structure data types used in monitor message data. The declarations are shown in the following programming languages:

- C
- COBOL
- PL/I
- RPG (ILE) (IBM i only)
- S/390 assembler (z/OS only)
- Visual Basic (Windows platforms only)
- "MQCFBS - Byte string parameter"
- "MQCFGR - Group parameter" on page 2849
- "MQCFH - PCF header" on page 2851
- "MQCFIL - Integer list parameter" on page 2855
- "MQCFIL64 - 64–bit integer list parameter" on page 2857
- "MQCFIN - Integer parameter" on page 2859
- "MQCFIN64 - 64–bit integer parameter" on page 2861
- "MQCFSL - String list parameter" on page 2863
- "MQCFST - String parameter" on page 2865
- "MQEPH - Embedded PCF header" on page 2868

MQCFBS - Byte string parameter

Use this page to view the structure of an MQCFBS parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, and S/390 assembler

The MQCFBS structure describes a byte string parameter. Following the links to the declarations is a description of the fields making up the MQCFBS structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- S/390 assembler-language (z/OS only)

Type

Description: This indicates that the structure is an MQCFBS structure describing a byte string parameter.
Data type: MQLONG.
Value:

MQCFT_BYTE_STRING
Structure defining a byte string.

StrucLength

Description: This is the length in bytes of the MQCFBS structure, including the variable-length string at the end of the structure (the *String* field).
Data type: MQLONG.

Parameter

Description: This identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.

StringLength

Description: This is the length in bytes of the data in the *String* field, and is zero or greater.
Data type: MQLONG.

String

Description: This is the value of the parameter identified by the *Parameter* field. The string is a byte string, and so is not subject to character-set conversion when sent between different systems.
Note: A null byte in the string is treated as normal data, and does not act as a delimiter for the string.
Data type: MQBYTE ×*StringLength*.

C language declaration

```
struct tagMQCFBS {
  MQLONG  Type;          /* Structure type */
  MQLONG  StrucLength;   /* Structure length */
  MQLONG  Parameter;     /* Parameter identifier */
  MQLONG  StringLength;  /* Length of string */
  MQBYTE  String[1];     /* String value -- first character */
 } MQCFBS;
```

COBOL language declaration

```
**   MQCFBS structure
  10 MQCFBS.
**    Structure type
   15 MQCFBS-TYPE         PIC S9(9) BINARY.
**    Structure length
   15 MQCFBS-STRUCLENGTH  PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFBS-PARAMETER    PIC S9(9) BINARY.
**    Length of string
   15 MQCFBS-STRINGLENGTH PIC S9(9) BINARY.
```

PL/I language declaration (z/OS only)

```
dcl
 1 MQCFBS based,
  3 Type         fixed bin(31), /* Structure type */
  3 StrucLength  fixed bin(31), /* Structure length */
  3 Parameter    fixed bin(31), /* Parameter identifier */
  3 StringLength fixed bin(31); /* Length of string */
```

RPG/ILE language declaration (IBM i only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFBS Structure
D*
D* Structure type
D  BSTYP                  1      4I 0 INZ(9)
D* Structure length
D  BSLEN                  5      8I 0 INZ(16)
D* Parameter identifier
D  BSPRM                  9     12I 0 INZ(0)
D* Length of string
D  BSSTL                 13     16I 0 INZ(0)
D* String value -- first byte
D  BSSRA                 17     17    INZ
```

S/390 assembler-language declaration (z/OS only)

```
MQCFBS               DSECT
MQCFBS_TYPE          DS   F  Structure type
MQCFBS_STRUCLENGTH   DS   F  Structure length
MQCFBS_PARAMETER     DS   F  Parameter identifier
MQCFBS_STRINGLENGTH  DS   F  Length of string
*
MQCFBS_LENGTH        EQU  *-MQCFBS
                     ORG  MQCFBS
MQCFBS_AREA          DS   CL(MQCFBS_LENGTH)
```

MQCFGR - Group parameter

Use this page to view the structure of an MQCFGR parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFGR structure describes a group parameter. Following the links to the declarations is a description of the fields making up the MQCFGR structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

The MQCFGR structure is a group parameter in which the subsequent parameter structures are grouped together as a single logical unit. The number of subsequent structures that are included is given by *ParameterCount*. This structure, and the parameter structures it includes, are counted as one structure only in the *ParameterCount* parameter in the PCF header (MQCFH) and the group parameter (MQCFGR).

Type

Description: Indicates that the structure type is MQCFGR describing which parameters are in this group.
Data type: MQLONG.
Value:

MQCFT_GROUP
Structure defining a group of parameters.

StrucLength

Description: Length in bytes of the MQCFGR structure.
Data type: MQLONG.
Value:

MQCFGR_STRUC_LENGTH
Length of the command format group-parameter structure.

Parameter

Description: This identifies the type of group parameter.
Data type: MQLONG.

ParameterCount

Description: The number of parameter structures following the MQCFGR structure that are contained within the group identified by the *Parameter* field. If the group itself contains one or more groups, each group and its parameters count as one structure only.
Data type: MQLONG.

C language declaration

```
typedef struct tagMQCFGR {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  ParameterCount;  /* Count of the grouped parameter structures */
 } MQCFGR;
```

COBOL language declaration

```
**   MQCFGR structure
  10 MQCFGR.
**    Structure type
   15 MQCFGR-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFGR-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFGR-PARAMETER      PIC S9(9) BINARY.
**    Count of grouped parameter structures
   15 MQCFGR-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (z/OS and Windows only)

```
dcl
 1 MQCFGR based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 ParameterCount fixed bin(31), /* Count of grouped parameter structures */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
  D* MQCFGR Structure
     D*
     D* Structure type
     D  GRTYP                  1      4I INZ(20)
     D* Structure length
     D  GRLEN                  5      8I INZ(16)
     D* Parameter identifier
     D  GRPRM                  9     12I INZ(0)
     D* Count of grouped parameter structures
     D  GRCNT                 13     16I INZ(0)
     D*
```

S/390 assembler-language declaration (z/OS only)

```
MQCFGR                  DSECT
MQCFGR_TYPE             DS   F        Structure type
MQCFGR_STRUCLENGTH      DS   F        Structure length
MQCFGR_PARAMETER        DS   F        Parameter identifier
MQCFGR_PARAMETERCOUNT   DS   F        Count of grouped parameter structures
MQCFGR_LENGTH           EQU  *-MQCFGR Length of structure
                        ORG  MQCFGR
MQCFGR_AREA             DS   CL(MQCFGR_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFGR
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  ParameterCount As Long  ' Count of grouped parameter structures
  End Type
```

MQCFH - PCF header

Use this page to view the structure of an MQCFH header and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFH structure describes the information that is present at the start of the message data of a monitoring message. Following the links to the declarations is a description of the fields making up the MQCFH structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- S/390 assembler language (z/OS only)
- Visual Basic language (Windows only)

Type

Description: Structure type This indicates the content of the message.
Data type: MQLONG.
Values:

MQCFT_ACCOUNTING
Message is an accounting message.

MQCFT_EVENT
Message is reporting an event.

MQCFT_REPORT
Message is an activity report.

MQCFT_RESPONSE
Message is a response to a command.

MQCFT_STATISTICS
Message is a statistics message.

MQCFT_TRACE_ROUTE
Message is a trace-route message.

StrucLength

Description: This is the length in bytes of the MQCFH structure
Data type: MQLONG.
Value:

MQCFH_STRUC_LENGTH
Length of command format header structure.

Version

Description: Structure version number.
Data type: MQLONG.
Value:

MQCFH_VERSION_1
Version number for all events except configuration and command events.

MQCFH_VERSION_2
Version number for configuration events.

MQCFH_VERSION_3
Version number for command events, activity reports, trace-route messages, accounting and statistics messages.

Command

Description: Specifies the category of the message.
Data type: MQLONG.
Value: Refer to the *Command* values in the following structure descriptions:

- "Event message MQCFH (PCF header)" on page 2913.
- Activity report MQCFH (PCF header) .
- Trace-route message MQCFH (PCF header).
- Message data in accounting and statistics messages.

MsgSeqNumber

Description: Message sequence number. This is the sequence number of the message within a set of related messages.
Data type: MQLONG.

Control

Description: Control options.
Data type: MQLONG.
Value:

MQCFC_LAST
Last message in the set.

MQCFC_NOT_LAST
Not the last message in the set.

CompCode

Description: Completion code.
Data type: MQLONG.
Value:

MQCC_OK
Events reporting OK condition, activity reports, trace-route messages, accounting messages, or statistics messages.

MQCC_WARNING
Event reporting warning condition.

Reason

Description: Reason code qualifying completion code.
Data type: MQLONG.
Value: For event messages:

MQRC_*
Dependent on the event being reported.
Note: Events with the same reason code are further identified by the *ReasonQualifier* parameter in the event data.

For activity reports, trace-route messages, accounting messages, and statistics messages:

MQRC_NONE

ParameterCount

Description: Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure.
Data type: MQLONG.
Value: 0 or greater.

C language declaration

```
typedef struct tagMQCFH {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Version;         /* Structure version number */
  MQLONG  Command;         /* Command identifier */
  MQLONG  MsgSeqNumber;    /* Message sequence number */
  MQLONG  Control;         /* Control options */
  MQLONG  CompCode;        /* Completion code */
  MQLONG  Reason;          /* Reason code qualifying completion code */
  MQLONG  ParameterCount;  /* Count of parameter structures */
 } MQCFH;
```

COBOL language declaration

```
**   MQCFH structure
  10 MQCFH.
**    Structure type
   15 MQCFH-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFH-STRUCLENGTH    PIC S9(9) BINARY.
**    Structure version number
   15 MQCFH-VERSION        PIC S9(9) BINARY.
**    Command identifier
   15 MQCFH-COMMAND        PIC S9(9) BINARY.
**    Message sequence number
   15 MQCFH-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Control options
   15 MQCFH-CONTROL        PIC S9(9) BINARY.
**    Completion code
```

```
   15 MQCFH-COMPCODE       PIC S9(9) BINARY.
**    Reason code qualifying completion code
   15 MQCFH-REASON         PIC S9(9) BINARY.
**    Count of parameter structures
   15 MQCFH-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (z/OS and Windows)

```
dcl
 1 MQCFH based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Version        fixed bin(31), /* Structure version number */
  3 Command        fixed bin(31), /* Command identifier */
  3 MsgSeqNumber   fixed bin(31), /* Message sequence number */
  3 Control        fixed bin(31), /* Control options */
  3 CompCode       fixed bin(31), /* Completion code */
  3 Reason         fixed bin(31), /* Reason code qualifying completion
                                     code */
  3 ParameterCount fixed bin(31); /* Count of parameter structures */
```

RPG language declaration (IBM i only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFH Structure
D*
D* Structure type
D  FHTYP                  1      4I 0 INZ(1)
D* Structure length
D  FHLEN                  5      8I 0 INZ(36)
D* Structure version number
D  FHVER                  9     12I 0 INZ(1)
D* Command identifier
D  FHCMD                 13     16I 0 INZ(0)
D* Message sequence number
D  FHSEQ                 17     20I 0 INZ(1)
D* Control options
D  FHCTL                 21     24I 0 INZ(1)
D* Completion code
D  FHCMP                 25     28I 0 INZ(0)
D* Reason code qualifying completion code
D  FHREA                 29     32I 0 INZ(0)
D* Count of parameter structures
D  FHCNT                 33     36I 0 INZ(0)
D*
```

S/390 assembler language declaration (z/OS only)

```
MQCFH                           DSECT
MQCFH_TYPE                      DS   F        Structure type
MQCFH_STRUCLENGTH               DS   F        Structure length
MQCFH_VERSION                   DS   F        Structure version number
MQCFH_COMMAND                   DS   F        Command identifier
MQCFH_MSGSEQNUMBER              DS   F        Message sequence number
MQCFH_CONTROL                   DS   F        Control options
MQCFH_COMPCODE                  DS   F        Completion code
MQCFH_REASON                    DS   F        Reason code qualifying
*                                             completion code
MQCFH_PARAMETERCOUNT            DS   F        Count of parameter
*                                             structures
MQCFH_LENGTH                    EQU  *-MQCFH  Length of structure
                                ORG  MQCFH
MQCFH_AREA                      DS   CL(MQCFH_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFH
  Type As Long            'Structure type
  StrucLength As Long     'Structure length
  Version As Long         'Structure version number
  Command As Long         'Command identifier
  MsgSeqNumber As Long    'Message sequence number
  Control As Long         'Control options
  CompCode As Long        'Completion code
  Reason As Long          'Reason code qualifying completion code
  ParameterCount As Long  'Count of parameter structures
End Type
```

MQCFIL - Integer list parameter

Use this page to view the structure of an MQCFIL parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFIL structure describes an integer list parameter. Following the links to the declarations is a description of the fields making up the MQCFIL structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

Type

Description: Indicates that the structure type is MQCFIL and describes an integer-list parameter.
Data type : MQLONG.
Value:

MQCFT_INTEGER_LIST
Structure defining an integer list.

StrucLength

Description: Length in bytes of the MQCFIL structure, including the array of integers at the end of the structure (the *values* field).
Data type : MQLONG.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type : MQLONG.

Count

Description: Number of elements in the *Values* array.
Data type : MQLONG.
Values: Zero or greater.

Values

Description: Array of values for the parameter identified by the *Parameter* field.
Data type : MQLONG×*Count*.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIL in a larger structure, and declare additional fields following MQCFIL, to represent the Values field as required.

C language declaration

```
typedef struct tagMQCFIL {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQLONG  Values[1];    /* Parameter values - first element */
 } MQCFIL;
```

COBOL language declaration

```
**   MQCFIL structure
  10 MQCFIL.
**    Structure type
   15 MQCFIL-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFIL based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31); /* Count of parameter values */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFIL Structure
     D*
     D* Structure type
     D  ILTYP                  1      4I 0
     D* Structure length
     D  ILLEN                  5      8I 0
     D* Parameter identifier
     D  ILPRM                  9     12I 0
     D* Count of paramter valuee
     D  ILCNT                 13     16I 0
```

S/390 assembler-language declaration

```
MQCFIL                  DSECT
MQCFIL_TYPE             DS   F        Structure type
MQCFIL_STRUCLENGTH      DS   F        Structure length
MQCFIL_PARAMETER        DS   F        Parameter identifier
MQCFIL_COUNT            DS   F        Count of parameter values
MQCFIL_LENGTH           EQU  *-MQCFIL Length of structure
                        ORG  MQCFIL
MQCFIL_AREA             DS   CL(MQCFIL_LENGTH)
```

Visual Basic language declaration

```
Type MQCFIL
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Count As Long        ' Count of parameter value
End Type
```

MQCFIL64 - 64–bit integer list parameter

Use this page to view the structure of an MQCFIL64 parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, and S/390 assembler

The MQCFIL64 structure describes a 64–bit integer list parameter. Following the links to the declarations is a description of the fields making up the MQCFIL64 structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)

Type

Description: Indicates that the structure is a MQCFIL64 structure describing a 64–bit integer list parameter.

Data type: MQLONG.

Value:

MQCFT_INTEGER64_LIST
Structure defining a 64–bit integer list.

StrucLength

Description: Length in bytes of the MQCFIL64 structure, including the array of integers at the end of the structure (the *Values* field).

Data type: MQLONG.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.

Count

Description: Number of elements in the *Values* array.
Data type: MQLONG.
Values: 0 or greater.

Values

Description: Array of values for the parameter identified by the *Parameter* field.
Data type: (MQINT64×*Count*)

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIL64 in a larger structure, and declare additional fields following MQCFIL64, to represent the *Values* field as required.

For COBOL, additional fields should be declared as:

```
PIC  S9(18)
```

For PL/I, additional fields should be declared as `FIXED BINARY SIGNED` with a precision of 63.

For System/390 assembler, additional fields should be declared `D` (double word) in the `DS` declaration.

C language declaration

```
typedef struct tagMQCFIN64 {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQINT64 Values[1];        /* Parameter value */
 } MQCFIL64;
```

COBOL language declaration

```
**   MQCFIL64 structure
  10 MQCFIL64.
**    Structure type
   15 MQCFIL64-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL64-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL64-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL64-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFIL64 based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31)  /* Count of parameter values */
```

RPG/ILE language declaration (IBM i only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFIL64 Structure
D*
D* Structure type
D  IL64TYP                1      4I 0 INZ(25)
D* Structure length
D  IL64LEN                5      8I 0 INZ(16)
D* Parameter identifier
D  IL64PRM                9     12I 0 INZ(0)
D* Count of parameter values
D  IL64CNT               13     16I 0 INZ(0)
D* Parameter values -- first element
D  IL64VAL               17     16    INZ(0)
```

S/390 assembler-language declaration (z/OS only)

```
MQCFIL64                        DSECT
MQCFIL64_TYPE                   DS   F        Structure type
MQCFIL64_STRUCLENGTH            DS   F        Structure length
MQCFIL64_PARAMETER              DS   F        Parameter identifier
MQCFIL64_COUNT                  DS   F        Parameter value high
MQCFIL64_LENGTH                 EQU  *-MQCFIL64 Length of structure
                               ORG  MQCFIL64
MQCFIL64_AREA                   DS   CL(MQCFIL64_LENGTH)
```

MQCFIN - Integer parameter

Use this page to view the structure of an MQCFIN parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFIN structure describes an integer parameter. Following the links to the declarations is a description of the fields making up the MQCFIN structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- S/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

Type

Description: Indicates that the structure type is MQCFIN and describes an integer parameter.
Data type: MQLONG.
Value:

MQCFT_INTEGER
Structure defining an integer.

StrucLength

Description: Length in bytes of the MQCFIN structure.
Data type: MQLONG.
Value:

MQCFIN_STRUC_LENGTH
Length of MQCFIN structure.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.

Value

Description: Value of parameter identified by the *Parameter* field.
Data type: MQLONG.

C language declaration

```
typedef struct tagMQCFIN {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Value;        /* Parameter value */
 } MQCFIN;
```

COBOL language declaration

```
**   MQCFIN structure
  10 MQCFIN.
**    Structure type
   15 MQCFIN-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIN-PARAMETER   PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN-VALUE       PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFIN based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Value       fixed bin(31); /* Parameter value */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFIN Structure
     D*
     D* Structure type
     D  INTYP                  1      4I 0
     D* Structure length
     D  INLEN                  5      8I 0
     D* Parameter identifier
     D  INPRM                  9     12I 0
     D* Parameter value
     D  INVAL                 13     16I 0
```

S/390 assembler-language declaration

```
MQCFIN                  DSECT
MQCFIN_TYPE             DS   F        Structure type
MQCFIN_STRUCLENGTH      DS   F        Structure length
MQCFIN_PARAMETER        DS   F        Parameter identifier
MQCFIN_VALUE            DS   F        Parameter value
MQCFIN_LENGTH           EQU  *-MQCFIN Length of structure
                        ORG  MQCFIN
MQCFIN_AREA             DS   CL(MQCFIN_LENGTH)
```

Visual Basic language declaration

```
Type MQCFIN
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Value As Long        ' Parameter value
End Type
```

MQCFIN64 - 64–bit integer parameter

Use this page to view the structure of an MQCFIN64 parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, and S/390 assembler

The MQCFIN64 structure describes a 64–bit integer parameter. Following the links to the declarations is a description of the fields making up the MQCFIN64 structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)

Type

Description: Indicates that the structure is a MQCFIN64 structure describing a 64–bit integer parameter.
Data type: MQLONG.
Value:

MQCFT_INTEGER64
Structure defining a 64–bit integer.

StrucLength

Description: Length in bytes of the MQCFIN64 structure.
Data type: MQLONG.
Value:

MQCFIN64_STRUC_LENGTH
Length of 64–bit integer parameter structure.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.

Values

Description: This is the value of the parameter identified by the *Parameter* field.
Data type: (MQINT64)

C language declaration

```
typedef struct tagMQCFIN64 {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Reserved;     /* Reserved */
  MQINT64 Value;        /* Parameter value */
 } MQCFIN64;
```

COBOL language declaration

```
**   MQCFIN64 structure
  10 MQCFIN64.
**    Structure type
   15 MQCFIN64-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN64-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIN64-PARAMETER   PIC S9(9) BINARY.
**    Reserved
   15 MQCFIN64-RESERVED    PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN64-VALUE       PIC S9(18) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFIN64 based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Reserved    fixed bin(31)  /* Reserved */
  3 Value       fixed bin(63); /* Parameter value */
```

RPG/ILE language declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFIN64 Structure
     D*
     D* Structure type
     D  IN64TYP                1      4I 0 INZ(23)
     D* Structure length
     D  IN64LEN                5      8I 0 INZ(24)
     D* Parameter identifier
     D  IN64PRM                9     12I 0 INZ(0)
     D* Reserved field
     D  IN64RSV               13     16I 0 INZ(0)
     D* Parameter value
     D  IN64VAL               17     16    INZ(0)
```

S/390 assembler-language declaration (z/OS only)

```
MQCFIN64                        DSECT
MQCFIN64_TYPE                   DS   F        Structure type
MQCFIN64_STRUCLENGTH            DS   F        Structure length
MQCFIN64_PARAMETER              DS   F        Parameter identifier
```

```
MQCFIN64_RESERVED               DS   F        Reserved
MQCFIN64_VALUE                  DS   D        Parameter value
MQCFIN64_LENGTH                 EQU  *-MQCFIN64 Length of structure
                              ORG  MQCFIN64
MQCFIN64_AREA                   DS   CL(MQCFIN64_LENGTH)
```

MQCFSL - String list parameter

Use this page to view the structure of an MQCFSL parameter and the declarations for the following programming languages: COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFSL structure describes a string list parameter. Following the links to the declarations is a description of the fields making up the MQCFSL structure:

- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

Type

Description: This indicates that the structure is an MQCFSL structure describing a string-list parameter.
Data type: MQLONG.
Value:

MQCFT_STRING_LIST
Structure defining a string list.

StrucLength

Description: This is the length in bytes of the MQCFSL structure, including the array of strings at the end of the structure (the *Strings* field).
Data type: MQLONG.

Parameter

Description: This identifies the parameter with values that are contained in the structure.
Data type: MQLONG.

CodedCharSetId

Description: This specifies the coded character set identifier of the data in the *Strings* field.
Data type: MQLONG.

Count

Description: This is the number of strings present in the *Strings* field; zero or greater.
Data type: MQLONG.

StringLength

Description: This is the length in bytes of one parameter value, that is the length of one string in the *Strings* field; all of the strings are this length.

Data type: MQLONG.

String

Description: This is a set of string values for the parameter identified by the *Parameter* field. The number of strings is given by the *Count* field, and the length of each string is given by the *StringLength* field. The strings are concatenated together, with no bytes skipped between adjacent strings. The total length of the strings is the length of one string multiplied by the number of strings present (that is, *StringLength*×*Count*).

In MQFMT_EVENT messages, trailing blanks can be omitted from string parameters (that is, the string may be shorter than the defined length of the parameter). *StringLength* gives the length of the string actually present in the message.
Note: In the MQCFSL structure, a null character in a string is treated as normal data, and does not act as a delimiter for the string. This means that when a receiving application reads a MQFMT_EVENT message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

Data type: MQCHAR × *StringLength*×*Count*.

COBOL language declaration

```
**   MQCFSL structure
  10 MQCFSL.
**    Structure type
   15 MQCFSL-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFSL-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFSL-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFSL-CODEDCHARSETID PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFSL-COUNT          PIC S9(9) BINARY.
**    Length of one string
   15 MQCFSL-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFSL based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 Count          fixed bin(31), /* Count of parameter values */
  3 StringLength   fixed bin(31); /* Length of one string */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFSL Structure
     D*
     D* Structure type
     D  SLTYP                  1      4I 0
     D* Structure length
     D  SLLEN                  5      8I 0
     D* Parameter identifier
     D  SLPRM                  9     12I 0
     D* Coded character set identifier
     D  SLCSI                 13     16I 0
```

```
     D* Count of parameter values
     D  SLCNT                17     20I 0
     D* Length of one string
     D  SLSTL                21     24I 0
```

S/390 assembler-language declaration (z/OS only)

```
MQCFSL                 DSECT
MQCFSL_TYPE            DS   F  Structure type
MQCFSL_STRUCLENGTH     DS   F  Structure length
MQCFSL_PARAMETER       DS   F  Parameter identifier
MQCFSL_CODEDCHARSETID  DS   F  Coded character set identifier
MQCFSL_COUNT           DS   F  Count of parameter values
MQCFSL_STRINGLENGTH    DS   F  Length of one string
*
MQCFSL_LENGTH          EQU  *-MQCFSL
                       ORG  MQCFSL
MQCFSL_AREA            DS   CL(MQCFSL_LENGTH)
```

Visual Basic language declaration (Windows systems only)

```
Type MQCFSL
  Type           As Long 'Structure type'
  StrucLength    As Long 'Structure length'
  Parameter      As Long 'Parameter identifier'
  CodedCharSetId As Long 'Coded character set identifier'
  Count          As Long 'Count of parameter values'
  StringLength   As Long 'Length of one string'
End Type
```

MQCFST - String parameter

Use this page to view the structure of an MQCFST parameter and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQCFST structure describes a string parameter. Following the links to the declarations is a description of the fields making up the MQCFST structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- System/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

The MQCFST structure ends with a variable-length character string; see the *String* field for further details.

Type

Description: Indicates that the structure type is MQCFST and describes a string parameter.
Data type: MQLONG.
Value:

MQCFT_STRING
Structure defining a string.

StrucLength

Description: Length in bytes of the MQCFST structure, including the string at the end of the structure (the *String* field).
Data type: MQLONG.

Parameter

Description: Identifies the parameter with a value that is contained in the structure.
Data type: MQLONG.
Values: Dependent on the event message.

CodedCharSetId

Description: Coded character set identifier of the data in the *String* field.
Data type: MQLONG.

StringLength

Description: Length in bytes of the data in the *String* field; zero or greater.
Data type: MQLONG.

String

Description: The value of the parameter identified by the *Parameter* field.

In MQFMT_EVENT messages, trailing blanks can be omitted from string parameters (that is, the string may be shorter than the defined length of the parameter). *StringLength* gives the length of the string actually present in the message.

Data type: MQCHAR×*StringLength*.
Value: The string can contain any characters that are in the character set defined by *CodedCharSetId*, and that are valid for the parameter identified by *Parameter*.
Language considerations: The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, System/390 assembler, and Visual Basic programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFST in a larger structure, and declare additional fields following MQCFST, to represent the *String* field as required.

A null character in the string is treated as normal data, and does not act as a delimiter for the string. This means that when a receiving application reads an MQFMT_EVENT message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

C language declaration

```
typedef struct tagMQCFST {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQLONG  StringLength;    /* Length of string */
  MQCHAR  String[1];       /* String value - first
                              character */
 } MQCFST;
```

COBOL language declaration

```
**   MQCFST structure
  10 MQCFST.
**    Structure type
   15 MQCFST-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFST-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFST-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFST-CODEDCHARSETID PIC S9(9) BINARY.
**    Length of string
   15 MQCFST-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration

```
dcl
 1 MQCFST based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 StringLength   fixed bin(31); /* Length of string */
```

RPG/ILE declaration (IBM i only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFST Structure
     D*
     D* Structure type
     D  STTYP                  1      4I 0
     D* Structure length
     D  STLEN                  5      8I 0
     D* Parameter identifier
     D  STPRM                  9     12I 0
     D* Coded character set identifier
     D  STCSI                 13     16I 0
     D* Length of string
     D  STSTL                 17     20I 0
```

S/390 assembler-language declaration

```
MQCFST                         DSECT
MQCFST_TYPE                    DS   F        Structure type
MQCFST_STRUCLENGTH             DS   F        Structure length
MQCFST_PARAMETER               DS   F        Parameter identifier
MQCFST_CODEDCHARSETID          DS   F        Coded character set
*                                            identifier
MQCFST_STRINGLENGTH            DS   F        Length of string
MQCFST_LENGTH                  EQU  *-MQCFST Length of structure
                               ORG  MQCFST
MQCFST_AREA                    DS   CL(MQCFST_LENGTH)
```

Visual Basic language declaration

```
Type MQCFST
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  CodedCharSetId As Long  ' Coded character set identifier
  StringLength As Long    ' Length of string
End Type
```

MQEPH - Embedded PCF header

Use this page to view the structure of an MQEPH embedded PCF header and the declarations for the following programming languages: C, COBOL, PL/I, RPG/ILE, S/390 assembler, and Visual Basic

The MQEPH structure describes the additional data that is present in a message when that message is a programmable command format (PCF) message. Following the links to the declarations is a description of the fields making up the MQEPH structure:

- C language
- COBOL language
- PL/I language (z/OS only)
- RPG/ILE language (IBM i only)
- S/390 assembler-language (z/OS only)
- Visual Basic language (Windows only)

The additional data consists of the MQEPH structure followed by an array of PCF parameter structures. To include the MQEPH structure in a message, the *Format* parameter in the message descriptor is set to MQFMT_EMBEDDED.

StrucId

Description: Structure identifier.
Data type: MQCHAR4.
Value:

MQEPH_STRUC_ID
Identifier for distribution header structure.

Version

Description: Structure version number.
Data type: MQLONG.
Value:

MQEPH_VERSION_1
Version number for embedded PCF header structure.

StrucLength

Description: Structure length. This is the length in bytes of the MQEPH structure and is set to the amount of data preceding the next header structure.
Data type: MQLONG.

Encoding

Description: Numeric encoding. This specifies the numeric encoding of the data that follows the last PCF parameter structure.
Data type: MQLONG.

CodedCharSetId

Description: Coded character set identifier. This specifies the coded character set identifier of the data that follows the last PCF parameter structure.
Data type: MQLONG.

Format

Description: Format. This specifies the format name of the data that follows the last PCF parameter structure.
Data type: MQCHAR8.

Flags

Description: Flags. This is a reserved field.
Data type: MQLONG.
Value:

MQEPH_NONE
No flags have been specified.

MQEPH_CCSID_EMBEDDED
The character set of the parameters containing character data is specified individually within the CodedCharSetId field in each structure. The character set of the StrucId and Format fields is defined by the CodedCharSetId field in the header structure that precedes the MQEPH structure, or by the CodedCharSetId field in the MQMD if the MQEPH is at the start of the message.

PCFHeader

Description: Command format header.
Data type: MQCFH.

C language declaration

```
struct tagMQEPH  {
  MQCHAR4 StrucId;         /* Structure identifier */
  MQLONG  Version;         /* Structure version number */
  MQLONG  StrucLength      /* Structure length */
  MQLONG  Encoding;        /* Numeric encoding */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQCHAR8 Format;          /* Data format */
  MQLONG  Flags;           /* Flags */
  MQCFH   PCFHeader;       /* PCF header */
 } MQEPH;
```

COBOL language declaration

```
**   MQEPH structure
  10 MQEPH.
**    Structure identifier
   15 MQEPH-STRUCID        PIC X(4).
**    Structure version number
   15 MQEPH-VERSION        PIC S9(9) BINARY.
**    Structure length
   15 MQEPH-STRUCLENGTH    PIC S9(9) BINARY.
**    Numeric encoding
   15 MQEPH-ENCODING       PIC S9(9) BINARY.
**    Coded characeter set identifier
   15 MQEPH-CODEDCHARSETID PIC S9(9) BINARY.
**    Data format
   15 MQEPH-FORMAT         PIC X(8).
**    Flags
   15 MQEPH-FLAGS          PIC S9(9) BINARY.
**    PCF header
```

```
   15 MQEPH-PCFHEADER.
**    Structure type
    20 MQEPH-PCFHEADER-TYPE            PIC S9(9) BINARY.
**    Structure length
    20 MQEPH-PCFHEADER-STRUCLENGTH     PIC S9(9) BINARY.
**    Structure version number
    20 MQEPH-PCFHEADER-VERSION         PIC S9(9) BINARY.
**    Command identifier
    20 MQEPH-PCFHEADER-COMMAND         PIC S9(9) BINARY.
**    Message sequence number
    20 MQEPH-PCFHEADER-MSGSEQNUMBER    PIC S9(9) BINARY.
**    Control options
    20 MQEPH-PCFHEADER-CONTROL         PIC S9(9) BINARY.
**    Completion code
    20 MQEPH-PCFHEADER-COMPCODE        PIC S9(9) BINARY.
**    Reason code qualifying completion code
    20 MQEPH-PCFHEADER-REASON          PIC S9(9) BINARY.
**    Count of parameter structures
    20 MQEPH-PCFHEADER-PARAMETERCOUNT  PIC S9(9) BINARY.
```

PL/I language declaration (z/OS and Windows)

```
dcl
 1 MQEPH based,
  3 StrucId         char(4),       /* Structure identifier */
  3 Version         fixed bin(31), /* Structure version number */
  3 StrucLength     fixed bin(31), /* Structure length */
  3 Encoding        fixed bin(31), /* Numeric encoding */
  3 CodedCharSetId  fixed bin(31), /* Coded character set identifier */
  3 Format          char(8),       /* Data format */
  3 Flags           fixed bin(31), /* Flags */
  3 PCFHeader,                     /* PCF header */
   5 Type           fixed bin(31), /* Structure type */
   5 StrucLength    fixed bin(31), /* Structure length */
   5 Version        fixed bin(31), /* Structure version number */
   5 Command        fixed bin(31), /* Command identifier */
   5 MsgSeqNumber   fixed bin(31), /* Message sequence number */
   5 Control        fixed bin(31), /* Control options */
   5 CompCode       fixed bin(31), /* Completion code */
   5 Reason         fixed bin(31), /* Reason code qualifying completion
                                      code */
   5 ParameterCount fixed bin(31); /* Count of parameter structures */
```

RPG language declaration (IBM i only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQEPH Structure
D*
D* Structure identifier
D  EPSID                  1      4    INZ('EPH ')
D* Structure version number
D  EPVER                  5      8I 0 INZ(1)
D* Structure length
D  EPLEN                  9     12I 0 INZ(68)
D* Numeric encoding
D  EPENC                 13     16I 0 INZ(0)
D* Coded character set identifier
D  EPCSI                 17     20I 0 INZ(0)
D* Format name
D  EPFMT                 21     28I 0 INZ('        ')
D* Flags
D  EPFLG                 29     32I 0 INZ(0)
D* Programmable Command Format Header
D*
D* Structure type
D  EP1TYPE               33     36I 0 INZ(0)
D* Structure length
D  EP1LEN                37     40I 0 INZ(36)
```

```
D* Structure version number
D  EP1VER                41     44I 0 INZ(3)
D* Command identifier
D  EP1CMD                45     48I 0 INZ(0)
D* Message sequence number
D  EP1SEQ                49     52I 0 INZ(1)
D* Control options
D  EP1CTL                53     56I 0 INZ(1)
D* Completion code
D  EP1CMP                57     60I 0 INZ(0)
D* Reason code qualifying completion code
D  EP1REA                61     64I 0 INZ(0)
D* Count of parameter structures
D  EP1CNT                65     68I 0 INZ(0)
```

S/390 assembler-language declaration (z/OS only)

```
MQEPH                          DSECT
MQEPH_STRUCID                  DS   CL4      Structure identifier
MQEPH_VERSION                  DS   F        Structure version number
MQEPH_STRUCLENGTH              DS   F        Structure length
MQEPH_ENCODING                 DS   F        Numeric encoding
MQEPH_CODEDCHARSETID           DS   F        Coded character set identifier
MQEPH_FORMAT                   DS   CL8      Data format
MQEPH_FLAGS                    DS   F        Flags
MQEPH_PCFHEADER                DS   0F       Force fullword alignment
MQEPH_PCFHEADER_TYPE           DS   F        Structure type
MQEPH_PCFHEADER_STRUCLENGTH    DS   F        Structure length
MQEPH_PCFHEADER_VERSION        DS   F        Structure version number
MQEPH_PCFHEADER_COMMAND        DS   F        Command identifier
MQEPH_PCFHEADER_MSGSEQNUMBER   DS   F        Message sequence number
MQEPH_PCFHEADER_CONTROL        DS   F        Control options
MQEPH_PCFHEADER_COMPCODE       DS   F        Completion code
MQEPH_PCFHEADER_REASON         DS   F        Reason code qualifying completion code
MQEPH_PCFHEADER_PARAMETERCOUNT DS   F        Count of parameter structures
MQEPH_PCFHEADER_LENGTH         EQU  *-MQEPH_PCFHEADER
                               ORG  MQEPH_PCFHEADER
MQEPH_PCFHEADER_AREA           DS   CL(MQEPH_PCFHEADER_LENGTH)
*
MQEPH_LENGTH                   EQU  *-MQEPH
                               ORG  MQEPH
MQEPH_AREA                     DS   CL(MQEPH_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQEPH
  StrucId As String*4     'Structure identifier
  Version As Long         'Structure version number
  StrucLength As Long     'Structure length
  Encoding As Long        'Numeric encoding
  CodedCharSetId As Long  'Coded characetr set identifier
  Format As String*8      'Format name
  Flags As Long           'Flags
  Reason As Long          'Reason code qualifying completion code
  PCFHeader As MQCFH      'PCF header
End Type
```

Object attributes for event data

Use this page to view the object attributes that WebSphere MQ monitoring techniques can include in the configuration event data recorded in event messages. The amount of event data depends on the type of object to which the configuration event relates.

Authentication information attributes

Event messages relating to objects can include authentication information attributes

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

AuthInfoConnName **(MQCFST)**
Authentication information connection name (parameter identifier: MQCA_AUTH_INFO_CONN_NAME).

The maximum length of the string is 48.

AuthInfoDesc **(MQCFST)**
Authentication information description (parameter identifier: MQCA_AUTH_INFO_DESC).

The maximum length of the string is MQ_AUTH_INFO_DESC_LENGTH.

AuthInfoType **(MQCFIN)**
Authentication information type (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value is MQAIT_CRL_LDAP.

LDAPPassword **(MQCFST)**
LDAP password (parameter identifier: MQCA_LDAP_PASSWORD).

The maximum length of the string is MQ_LDAP_PASSWORD_LENGTH.

LDAPUserName **(MQCFST)**
LDAP user name (parameter identifier: MQCA_LDAP_USER_NAME).

The maximum length of the string is 256.

CF structure attributes

Event messages relating to objects can include CF structure attributes

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

CFLevel **(MQCFIN)**
CF level (parameter identifier: MQIA_CF_LEVEL).

CFStrucDesc **(MQCFST)**
CF Structure description (parameter identifier: MQCA_CF_STRUC_DESC).

The maximum length of the string is MQCA_CF_STRUC_DESC_LENGTH.

Recovery **(MQCFIN)**
Recovery (parameter identifier: MQIA_CF_RECOVER).

Communication information attributes

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss`.

Bridge **(MQCFIN)**
Bridge (parameter identifier: MQIA_MCAST_BRIDGE).

Specifies whether publications from applications not using Multicast are bridged to applications using multicast.

The value can be:

MQMCB_DISABLED
Bridging is disabled.

MQMCB_ENABLED
Bridging is enabled.

CCSID **(MQCFIN)**
Coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

The CCSID that messages are transmitted on.

CommEvent **(MQCFIN)**
Communication event (parameter identifier: MQIA_COMM_EVENT).

Controls whether event messages are generated for multicast handles that are created using this COMMINFO object.

The value can be:

MQEVR_DISABLED
Event messages are not generated.

MQEVR_ENABLED
Event messages are generated.

MQEVR_EXCEPTION
Event messages are generated if the message reliability is below the reliability threshold.

ComminfoName **(MQCFST)**
Communication information name (parameter identifier: MQCA_COMM_INFO_NAME).

The name of the administrative communication information definition about which information is to be returned.

Description **(MQCFST)**
Description (parameter identifier: MQCA_COMM_INFO_DESC).

Plain-text comment that provides descriptive information about the communication information object.

Encoding **(MQCFIN)**
Encoding (parameter identifier: MQIACF_ENCODING).

The encoding that the messages are transmitted in.

The value can be:

MQENC_AS_PUBLISHED

MQENC_NORMAL

MQENC_REVERSED

MQENC_S390

MQENC_TNS

GrpAddress **(MQCFST)**
Group address (parameter identifier: MQCACH_GROUP_ADDRESS).

The group IP address or DNS name.

MonitorInterval **(MQCFIN)**
Frequency of monitoring (parameter identifier: MQIA_MONITOR_INTERVAL).

How frequently, in seconds, monitoring information is updated and event messages are generated.

MulticastHeartbeat **(MQCFIN)**
Multicast heartbeat (parameter identifier: MQIACH_MC_HB_INTERVAL).

Heartbeat interval measured in milliseconds.

MulticastPropControl **(MQCFIN)**
Multicast properties control (parameter identifier: MQIACH_MULTICAST_PROPERTIES).

Controls how many of the MQMD properties and user properties flow with the message.

The value can be:

MQMCP_ALL
All properties are transmitted.

MQMCP_REPLY
Only user properties and MQMD fields that deal with replying to the messages are transmitted.

MQMCP_USER
Only user properties are transmitted.

MQMCP_NONE
No properties are transmitted.

MQMCP_COMPAT
Properties are transmitted in a format compatible with previous WebSphere MQ multicast clients.

MsgHistory **(MQCFIN)**
Message history (parameter identifier: MQIACH_MSG_HISTORY).

The amount of message history in kilobytes that is kept by the system to handle retransmissions in the case of NACKs.

NewSubHistory **(MQCFIN)**
New Subscriber History (parameter identifier: MQIACH_NEW_SUBSCRIBER_HISTORY).

Controls how much historical data a new subscriber receives. The value can be:

MQNSH_NONE
Only publications from the time of the subscription are sent.

MQNSH_ALL
As much history as is known is retransmitted.

PortNumber **(MQCFIN)**
Port Number (parameter identifier: MQIACH_PORT).

The port number to transmit on.

Type **(MQCFIN)**
Type (parameter identifier: MQIA_COMM_INFO_TYPE).

The type of the communications information object.

Channel attributes

Event messages relating to objects can include channel attributes

Only those attributes that apply to the type of channel in question are included in the event data.

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

BatchHeartbeat **(MQCFIN)**
The value being used for the batch heartbeating (parameter identifier: MQIACH_BATCH_HB).

The value can be in the range 0 through 999999. A value of 0 indicates heartbeating is not in use.

BatchInterval **(MQCFIN)**
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

BatchSize **(MQCFIN)**
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

ChannelDesc **(MQCFST)**
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

ChannelMonitoring **(MQCFIN)**
Level of monitoring data collection for the channel (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_OFF
Monitoring data collection is turned off.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a medium ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

MQMON_Q_MGR
The level of monitoring data collected is based on the queue manager attribute *ChannelMonitoring*.

ChannelName **(MQCFST)**
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType **(MQCFIN)**
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value can be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

CipherSpec **(MQCFST)**
SSL cipher specification (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The maximum length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterNamelist **(MQCFST)**
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

CLWLChannelPriority **(MQCFIN)**
Cluster workload channel priority (parameter identifier: MQIACH_CLWL_CHANNEL_PRIORITY).

CLWLChannelRank **(MQCFIN)**
Cluster workload channel rank (parameter identifier: MQIACH_CLWL_CHANNEL_RANK).

CLWLChannelWeight **(MQCFIN)**
Cluster workload channel weight (parameter identifier: MQIACH_CLWL_CHANNEL_WEIGHT).

ConnectionName **(MQCFST)**
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

DataConversion **(MQCFIN)**
Whether sender should convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

DiscInterval **(MQCFIN)**
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

HeaderCompression **(MQCFIL)**
Header data compression techniques supported by the channel (parameter identifier: MQIACH_HDR_COMPRESSION).

For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of the following:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval **(MQCFIN)**
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

KeepAliveInterval **(MQCFIN)**
Keep alive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

LocalAddress **(MQCFST)**
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetryCount **(MQCFIN)**
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

LongRetryInterval **(MQCFIN)**
Long timer (parameter identifier: MQIACH_LONG_TIMER).

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

MCAName **(MQCFST)**
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

The maximum length of the string is MQ_MCA_NAME_LENGTH.

MCAType **(MQCFIN)**
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

The value can be:

MQMCAT_PROCESS
Process

MQMCAT_THREAD
Thread

MCAUserIdentifier **(MQCFST)**
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

The maximum length of the MCA user identifier is MQ_MCA_USER_ID_LENGTH.

MessageCompression **(MQCFIL)**
Message data compression techniques supported by the channel (parameter identifier: MQIACH_MSG_COMPRESSION).

For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of:

MQCOMPRESS_NONE
No message data compression is performed. This is the default value.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_ANY
Any compression technique supported by the queue manager can be used. This is only valid for receiver, requester, and server-connection channels.

ModeName **(MQCFST)**
Mode name (parameter identifier: MQCACH_MODE_NAME).

The maximum length of the string is MQ_MODE_NAME_LENGTH.

MsgExit **(MQCFSL)**
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the *Count* for *MsgUserData*. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

MsgRetryCount **(MQCFIN)**
Message retry count (parameter identifier: MQIACH_MR_COUNT).

Specifies the number of times that a failing message should be retried.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

MsgRetryExit **(MQCFST)**
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

This parameter is only valid for receiver, cluster-receiver, and requester channels.

The maximum length of the string is MQ_MAX_EXIT_NAME_LENGTH.

MsgRetryInterval **(MQCFIN)**
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

Specifies the minimum time interval in milliseconds between retries of failing messages.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

MsgRetryUserData **(MQCFST)**
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

Specifies user data that is passed to the message retry exit.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgUserData **(MQCFSL)**
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the count for *MsgExit*. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

NetworkPriority **(MQCFIN)**
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

NonPersistentMsgSpeed **(MQCFIN)**
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password **(MQCFST)**
Password (parameter identifier: MQCACH_PASSWORD).

The maximum length of the string is MQ_PASSWORD_LENGTH.

PeerName **(MQCFST)**
SSL peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The maximum length of the string is 256.

PutAuthority **(MQCFIN)**
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

The value can be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

MQPA_ALTERNATE_OR_MCA
Alternate or MCA user identifier is used.

MQPA_ONLY_MCA
Only MCA user identifier is used.

QMgrName **(MQCFST)**
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReceiveExit **(MQCFSL)**
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the *Count* for *ReceiveUserData*. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the *StringLength* field in that structure.

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

ReceiveUserData **(MQCFSL)**
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the count for *ReceiveExit*. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SecurityExit **(MQCFST)**
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

SecurityUserData **(MQCFST)**
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendExit **(MQCFSL)**
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the *Count* for *SendUserData*. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the *StringLength* field in that structure.

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

SendUserData **(MQCFSL)**
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the count for *SendExit*. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SeqNumberWrap **(MQCFIN)**
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

ShortRetryCount **(MQCFIN)**
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

ShortRetryInterval **(MQCFIN)**
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

SSLClientAuthentication **(MQCFIN)**
SSL client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Certificate required.

MQSCA_OPTIONAL
Certificate optional.

TpName **(MQCFST)**
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The maximum length of the string is MQ_TP_NAME_LENGTH.

TransportType **(MQCFIN)**
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value may be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

UserIdentifier **(MQCFST)**
Task user identifier (parameter identifier: MQCACH_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

XmitQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Channel authentication attributes

Event messages relating to objects can include channel authentication attributes

Only those attributes that apply to the type of channel in question are included in the event data.

ChannelProfile **(MQCFST).**
Channel Profile (parameter identifier: MQCACH_CHANNEL_NAME).

Maximum length is MQ_CHANNEL_NAME_LENGTH.

Returned: Always.

ChannelAuthType **(MQCFIN).**
Channel Authentication Type (parameter identifier: MQIACF_CHLAUTH_TYPE).

Returned: Always.

Warning **(MQCFIN).**
Warning (parameter identifier: MQIACH_WARNING).

Returned: Always.

connectionNameList **(MQCFSL).**
Connection Name List (parameter identifier: MQCACH_CONNECTION_NAME_LIST).

Element length: MQ_CONN_NAME_LENGTH.

Returned: Only when Channel Auth Type is MQAUT_BLOCKADDR.

MCAUserIdList **(MQCFSL).**
MCA User Id List (parameter identifier: MQCACH_MCA_USER_ID_LIST).

Element length: MQ_MCA_USER_ID_LENGTH

Returned: Only when Channel Auth Type is MQAUT_BLOCKUSER

MCAUser **(MQCFST).**
MCA User (parameter identifier: MQCACH_MCA_USER_ID).

Maximum length: MQ_MCA_USER_ID_LENGTH.

Returned: Only when Channel Auth Type is of a mapping type (MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP or MQCAUT_QMGRMAP).

ConnectionName **(MQCFST).**
Connection Name (parameter identifier: MQCACH_CONNECTION_NAME).

Maximum length: MQ_CONN_NAME_LENGTH

Returned: Only when Channel Auth Type is of a mapping type (MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP or MQCAUT_QMGRMAP).

UserSource **(MQCFIN).**
User Source (parameter identifier: MQIACH_USER_SOURCE).

Returned: Only when Channel Auth Type is of a mapping type (MQCAUT_SSLPEERMAP, MQCAUT_ADDRESSMAP, MQCAUT_USERMAP or MQCAUT_QMGRMAP).

SSLPeerName **(MQCFST).**
SSL Peer Name (parameter identifier: MQCACH_SSL_PEER_NAME).

Maximum length: MQ_SSL_PEER_NAME_LENGTH.

Returned: Only when Channel Auth Type is MQCAUT_SSLPEERMAP.

ClientUserId **(MQCFST).**
Client User Id (parameter identifier: MQCACH_CLIENT_USER_ID).

Maximum length: MQ_MCA_USER_ID_LENGTH.

Returned: Only when Channel Auth Type is MQCAUT_USERMAP.

RemoteQueueManagerName **(MQCFST).**
Remote Queue Manager Name (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

Maximum length: MQ_Q_MGR_NAME_LENGTH.

Returned: Only when Channel Auth Type is MQCAUT_QMGRMAP.

Listener attributes

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date, in the form `yyyy-mm-dd`, on which the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time, in the form `hh.mm.ss`, at which the information was last altered.

Adapter **(MQCIN)**
Adapter number (parameter identifier: MQIACH_ADAPTER).

The adapter number on which NetBIOS listens. This parameter is valid only on Windows.

Backlog **(MQCIN)**
Backlog (parameter identifier: MQIACH_BACKLOG).

The number of concurrent connection requests that the listener supports.

Commands **(MQCIN)**
Adapter number (parameter identifier: MQIACH_COMMAND_COUNT).

The number of commands that the listener can use. This parameter is valid only on Windows.

IPAddress **(MQCFST)**
IP address (parameter identifier: MQCACH_IP_ADDRESS).

IP address for the listener specified in IPv4 dotted decimal, IPv6 hexadecimal notation, or alphanumeric host name form.

ListenerDesc **(MQCFST)**
Description of listener definition (parameter identifier: MQCACH_LISTENER_DESC).

ListenerName **(MQCFST)**
Name of listener definition (parameter identifier: MQCACH_LISTENER_NAME).

LocalName **(MQCFST)**
NetBIOS local name (parameter identifier: MQCACH_LOCAL_NAME).

The NetBIOS local name that the listener uses. This parameter is valid only on Windows.

NetbiosNames **(MQCFIN)**
NetBIOS names (parameter identifier: MQIACH_NAME_COUNT).

The number of names that the listener supports. This parameter is valid only on Windows.

Port **(MQCFIN)**
Port number (parameter identifier: MQIACH_PORT).

The port number for TCP/IP. This parameter is valid only if the value of TransportType is MQXPT_TCP.

Sessions **(MQCFIN)**
NetBIOS sessions (parameter identifier: MQIACH_SESSION_COUNT).

The number of sessions that the listener can use. This parameter is valid only on Windows.

Socket **(MQCFIN)**
SPX socket number (parameter identifier: MQIACH_SOCKET).

The SPX socket on which to listen. This parameter is valid only if the value of TransportType is MQXPT_SPX.

StartMode **(MQCFIN)**
Service mode (parameter identifier: MQIACH_LISTENER_CONTROL).

Specifies how the listener is to be started and stopped. The value can be:

MQSVC_CONTROL_MANUAL
The listener is started and stopped manually, by user command.

MQSVC_CONTROL_Q_MGR
The listener is started and stopped when the queue manager starts and stops.

MQSVC_CONTROL_Q_MGR_START
The listener is started when the queue manager starts, but does not stop when the queue manager stops.

TPName **(MQCFST)**
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The LU 6.2 transaction program name. This parameter is valid only on Windows.

TransportType **(MQCFIN)**
Transmission protocol (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_TCP
TCP

MQXPT_LU62
LU 6.2

MQXPT_NETBIOS
NetBIOS

MQXPT_SPX
SPX

Namelist attributes

Event messages relating to objects can include namelist attributes

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

NameCount **(MQCFIN)**
Number of names in the namelist (parameter identifier: MQIA_NAME_COUNT).

The number of names contained in the namelist.

NamelistDesc **(MQCFST)**
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

NamelistName **(MQCFST)**
The name of the namelist definition (parameter identifier: MQCA_NAMELIST_NAME).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

NamelistType **(MQCFIN)**
Namelist type (parameter identifier: MQIA_NAMELIST_TYPE).

Names **(MQCFSL)**
The names contained in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the *Count* field in the MQCFSL structure. The length of each name is given by the *StringLength* field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

Process attributes

Event messages relating to objects can include process attributes

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

ApplId **(MQCFST)**
Application identifier (parameter identifier: MQCA_APPL_ID).

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

ApplType **(MQCFIN)**
Application type (parameter identifier: MQIA_APPL_TYPE).

EnvData **(MQCFST)**
Environment data (parameter identifier: MQCA_ENV_DATA).

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc **(MQCFST)**
Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

ProcessName **(MQCFST)**
The name of the process definition (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

UserData **(MQCFST)**
User data (parameter identifier: MQCA_USER_DATA).

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Queue attributes

Event messages relating to objects can include queue attributes

Only those attributes that apply to the type of queue in question are included in the event data.

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

BackoutRequeueName **(MQCFST)**
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

BackoutThreshold **(MQCFIN)**
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

BaseQName **(MQCFST)**
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This is the name of a queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CFstructure **(MQCFST)**
CF structure name (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

ClusterName **(MQCFST)**
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterNamelist **(MQCFST)**
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

CLWLQueuePriority **(MQCFIN)**
Queue priority (parameter identifier: MQIA_CLWL_Q_PRIORITY).

CLWLQueueRank **(MQCFIN)**
Queue rank (parameter identifier: MQIA_CLWL_Q_RANK).

CLWLUseQ **(MQCFIN)**
This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance (parameter identifier: MQIA_CLWL_USEQ).

The value can be:

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

MQCLWL_USEQ_AS_Q_MGR
Inherit definition from the queue manager attribute *CLWLUseQ*.

CreationDate **(MQCFST)**
Queue creation date (parameter identifier: MQCA_CREATION_DATE).

The maximum length of the string is MQ_CREATION_DATE_LENGTH.

CreationTime **(MQCFST)**
Creation time (parameter identifier: MQCA_CREATION_TIME).

The maximum length of the string is MQ_CREATION_TIME_LENGTH.

DefBind **(MQCFIN)**
Default binding (parameter identifier: MQIA_DEF_BIND).

The value can be:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

MQBND_BIND_ON_GROUP
Allows an application to request that a group of messages are all allocated to the same destination instance.

DefinitionType **(MQCFIN)**
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

The value can be:

MQQDT_PREDEFINED
Predefined permanent queue.

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_SHARED_DYNAMIC
Dynamically defined permanent queue that is shared.

DefInputOpenOption **(MQCFIN)**
Default input open option for defining whether queues can be shared (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

The value can be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

DefPersistence **(MQCFIN)**
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

The value can be:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority **(MQCFIN)**
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

HardenGetBackout **(MQCFIN)**
Whether to harden backout (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

The value can be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count may not be remembered.

IndexType **(MQCFIN)**
Index type (parameter identifier: MQIA_INDEX_TYPE).

InhibitGet **(MQCFIN)**
Whether get operations are allowed (parameter identifier: MQIA_INHIBIT_GET).

The value can be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut **(MQCFIN)**
Whether put operations are allowed (parameter identifier: MQIA_INHIBIT_PUT).

The value can be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

InitiationQName **(MQCFST)**
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

MaxQDepth **(MQCFIN)**
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

MsgDeliverySequence **(MQCFIN)**
Whether priority is relevant (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value can be:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

ProcessName **(MQCFST)**
Name of process definition for queue (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

QDepthHiEvent **(MQCFIN)**
Controls whether Queue Depth High events are generated. (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

The value can be:

MQEVR_ENABLED
Queue depth high events are enabled.

MQEVR_DISABLED
Queue depth high events are disabled.

QDepthHighLimit **(MQCFIN)**
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth High event.

QDepthLoEvent **(MQCFIN)**
Controls whether Queue Depth Low events are generated. (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

The value can be:

MQEVR_ENABLED
Queue depth lowevents are enabled.

MQEVR_DISABLED
Queue depth low events are disabled.

QDepthLowLimit **(MQCFIN)**
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

QDepthMaxEvent **(MQCFIN)**
Controls whether Queue Full events are generated. (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

The value can be:

MQEVR_ENABLED
Queue depth full events are enabled.

MQEVR_DISABLED
Queue depth full events are disabled.

QDesc **(MQCFST)**
Queue description (parameter identifier: MQCA_Q_DESC).

The maximum length of the string is MQ_Q_DESC_LENGTH.

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QServiceInterval **(MQCFIN)**
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events.

QType **(MQCFIN)**
Queue type (parameter identifier: MQIA_Q_TYPE).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

QueueAccounting **(MQCFIN)**
Specifies whether accounting information is collected (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_ON
Accounting information is collected for the queue.

MQMON_OFF
Accounting information is not collected for the queue.

MQMON_Q_MGR
The collection of accounting information for this queue is based is based on the queue manager attribute *QueueAccounting*.

QueueMonitoring **(MQCFIN)**
Level of monitoring data collection for the queue (parameter identifier: MQIA_MONITORING_Q).

The value can be:

MQMON_OFF
Monitoring data collection is turned off.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

MQMON_Q_MGR
The level of monitoring data collected is based on the queue manager attribute *QueueMonitoring*.

RemoteQMgrName **(MQCFST)**
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

RemoteQName **(MQCFST)**
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

RetentionInterval **(MQCFIN)**
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

ServiceIntervalEvent **(MQCFIN)**
Controls whether Service Interval High or Service Interval OK events are generated. .

The value can be:

MQQSIE_NONE
No service interval events are generated.

MQQSIE_OK
Service interval OK events are generated.

MQQSIE_HIGH
Service interval high events are generated.

Shareability **(MQCFIN)**
Whether queue can be shared (parameter identifier: MQIA_SHAREABILITY).

The value can be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

StorageClass **(MQCFST)**
Storage class name (parameter identifier: MQCA_STORAGE_CLASS).

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

TriggerControl **(MQCFIN)**
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

The value can be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerData **(MQCFST)**
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

TriggerDepth **(MQCFIN)**
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

TriggerMsgPriority **(MQCFIN)**
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

TriggerType **(MQCFIN)**
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

The value can be:

MQTT_NONE
No trigger messages.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_EVERY
Trigger message for every message.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

Usage **(MQCFIN)**
Usage (parameter identifier: MQIA_USAGE).

The value can be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

XmitQName **(MQCFST)**
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue manager attributes

Event messages relating to objects can include queue manager attributes.

ActivityRecording **(MQCFIN)**
Specifies whether activity recording is enabled or disabled (parameter identifier: MQIA_ACTIVITY_RECORDING).

The value can be:

MQRECORDING_MSG
Activity recording is enabled. Activity reports are delivered to the reply-to queue specified in the message descriptor of the message.

MQRECORDING_Q
Activity recording is enabled. Activity reports are delivered to a fixed name queue.

MQRECORDING_DISABLED.
Activity recording is disabled.

AdoptNewMCACheck **(MQCFIN)**
Procedure to determine if an existing receiver MCA is to be adopted when an inbound channel is detected of the same name (parameter identifier: MQIA_ADOPTNEWMCA_CHECK).

The value can be:

MQADOPT_CHECK_Q_MGR_NAME
Compare the receiver MCA and the inbound channel. If the queue manager names match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is canceled, and a new MCA is created.

MQADOPT_CHECK_NET_ADDR
Compare the receiver MCA and the inbound channel. If the network addresses match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is canceled, and a new MCA is created.

MQADOPT_CHECK_ALL
Compare the receiver MCA and the inbound channel. If both the queue manager names, and the network addresses match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is canceled, and a new MCA is created.

MQADOPT_CHECK_NONE
If the existing receiver MCA is active it is adopted with no checks.

AdoptNewMCAType **(MQCFIN)**
Specifies whether orphaned receiver MCAs are to be restarted when an inbound channel matching the *AdoptNewMCACheck* procedure is detected (parameter identifier: MQIA_ADOPTNEWMCA_TYPE).

The value can be:

MQADOPT_TYPE_NO
Do not restart and adopt orphaned receiver MCAs.

MQADOPT_TYPE_ALL
Restart and adopt orphaned receiver MCAs.

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

AuthorityEvent **(MQCFIN)**
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

BridgeEvent **(MQCFIN)**
Determines whether IMS bridge events are generated (parameter identifier: MQIA_BRIDGE_EVENT).

The value can be:

MQEVR_ENABLED
All IMS bridge events are enabled.

MQEVR_DISABLED
All IMS bridge events are disabled.

ChannelAuthenticationRecords **(MQCFIN)**
Controls whether channel authentication records are used (parameter identifier: MQIA_CHLAUTH_RECORDS).

Channel authentication records can be set and displayed regardless of the value of this attribute.

The value can be any of the following values:

MQCHLA_DISABLED
Channel authentication records are not checked.

MQCHLA_ENABLED
Channel authentication records are checked.

ChannelAutoDefExit **(MQCFST)**
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ChannelEvent **(MQCFIN)**
Determines whether channel events are generated (parameter identifier: MQIA_CHANNEL_EVENT).

The value can be:

MQEVR_ENABLED
All channel events are enabled.

MQEVR_EXCEPTION
Only the following channels events are enabled:

- MQRC_CHANNEL_ACTIVATED
- MQRC_CHANNEL_CONV_ERROR
- MQRC_CHANNEL_NOT_ACTIVATED
- MQRC_CHANNEL_STOPPED

MQEVR_DISABLED
All channel events are disabled.

ChannelMonitoring **(MQCFIN)**
Level of real-time monitoring data collection for channels (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_NONE
Monitoring data collection is disabled, regardless of the setting for the *ChannelMonitoring* channel attribute.

MQMON_OFF
Monitoring data collection is turned off for channels specifying MQMON_Q_MGR in the *ChannelMonitoring* channel attribute.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection for channels specifying MQMON_Q_MGR in the *ChannelMonitoring* channel attribute.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection for channels specifying MQMON_Q_MGR in the *ChannelMonitoring* channel attribute.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection for channels specifying MQMON_Q_MGR in the *ChannelMonitoring* channel attribute.

ChinitAdapters **(MQCFIN)**
Number of channel initiator adapter subtasks to use for processing WebSphere MQ calls (parameter identifier: MQIA_CHINIT_ADAPTERS).

This value must be in the range 0 through 9999.

ChinitDispatchers **(MQCFIN)**
Number of dispatchers to use for the channel initiator (parameter identifier: MQIA_CHINIT_DISPATCHERS).

ChinitServiceParm **(MQCFST)**
This attribute is reserved for use by IBM (parameter identifier: MQCA_CHINIT_SERVICE_PARM).

ChinitTraceAutoStart **(MQCFIN)**
Specifies whether the channel initiator trace should start automatically (parameter identifier: MQIA_CHINIT_TRACE_AUTO_START).

The value can be:

MQTRAXSTR_YES
Channel initiator trace starts automatically.

MQTRAXSTR_NO
Channel initiator trace does not starts automatically.

ChinitTraceTableSize **(MQCFIN)**
Size of the channel initiator's trace data space, in MB (parameter identifier: MQIA_CHINIT_TRACE_TABLE_SIZE).

ClusterSenderMonitoring **(MQCFIN)**
Level of real-time monitoring data collection for auto-defined cluster sender channels (parameter identifier: MQIA_MONITORING_AUTO_CLUSSDR).

The value can be:

MQMON_Q_MGR
The collection of monitoring data is inherited from the setting of the *ChannelMonitoring* attribute in the queue manager object.

MQMON_OFF
Monitoring data collection is disabled.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

ClusterWorkLoadData **(MQCFST)**
Data passed to the cluster workload exit (parameter identifier: MQCA_CLUSTER_WORKLOAD_DATA).

ClusterWorkLoadExit **(MQCFST)**
Name of the cluster workload exit (parameter identifier: MQCA_CLUSTER_WORKLOAD_EXIT).

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

ClusterWorkLoadLength **(MQCFIN)**
Cluster workload length (parameter identifier: MQIA_CLUSTER_WORKLOAD_LENGTH).

The maximum length of the message passed to the cluster workload exit.

CLWLMRUChannels **(MQCFIN)**
Maximum number of most recently used channels for cluster workload balancing (parameter identifier: MQIA_CLWL_MRU_CHANNELS).

CLWLUseQ **(MQCFIN)**
This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance (parameter identifier: MQIA_CLWL_USEQ).

The value can be:

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CodedCharSetId **(MQCFIN)**
Coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

CommandEvent **(MQCFIN)**
Controls whether command events are generated (parameter identifier: MQIA_COMMAND_EVENT).

The value can be:

MQEVR_DISABLED
Command event generation disabled.

MQEVR_ENABLED
Command event generation enabled.

MQEVR_NO_DISPLAY
Command events are generated for all commands other than MQSC DISPLAY commands and PCF Inquire commands.

CommandInputQName **(MQCFST)**
Command input queue name (parameter identifier: MQCA_COMMAND_INPUT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

CommandLevel **(MQCFIN)**
Command level supported by queue manager (parameter identifier: MQIA_COMMAND_LEVEL).

ConfigurationEvent **(MQCFIN)**
Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT).

The value can be:

MQEVR_DISABLED
Configuration event generation disabled.

MQEVR_ENABLED
Configuration event generation enabled.

CPILevel **(MQCFIN)**
CPI level (parameter identifier: MQIA_CPI_LEVEL).

DeadLetterQName **(MQCFST)**
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DefXmitQName **(MQCFST)**
Default transmission queue name (parameter identifier: MQCA_DEF_XMIT_Q_NAME).

This is the name of the default transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DNSGroup **(MQCFST)**
The name of the group that the TCP listener that handles inbound transmissions for the queue sharing group must join when using Workload Manager for Dynamic Domain Name Services (parameter identifier: MQCA_DNS_GROUP).

The maximum length of this name is MQ_DNS_GROUP_NAME_LENGTH.

DNSWLM **(MQCFIN)**
Specifies whether the TCP listener that handles inbound transmissions for the queue sharing group will register with the Workload Manager for Dynamic Domain Name Services (parameter identifier: MQIA_DNS_WLM).

The value can be:

MQDNSWLM_YES
Register with the Workload Manager for Dynamic Domain Name Services.

MQDNSWLM_NO
Do not register with the Workload Manager for Dynamic Domain Name Services.

ExpiryInterval **(MQCFIN)**
Expiry interval (parameter identifier: MQIA_EXPIRY_INTERVAL).

GroupUR **(MQCFIN)**
Controls whether XA client applications can establish transactions with a GROUP unit of recovery disposition.

The value can be:

MQGUR_DISABLED
XA client applications must connect using a queue manager name.

MQGUR_ENABLED
XA client applications can establish transactions with a group unit of recovery disposition by specifying a QSG name when they connect.

IGQPutAuthority **(MQCFIN)**
IGQ put authority (parameter identifier: MQIA_IGQ_PUT_AUTHORITY).

IGQUserId **(MQCFST)**
IGQ user identifier (parameter identifier: MQCA_IGQ_USER_ID).

The maximum length of the **string** is MQ_USER_ID_LENGTH.

InhibitEvent **(MQCFIN)**
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

IntraGroupQueueing **(MQCFIN)**
Intra group queueing (parameter identifier: MQIA_INTRA_GROUP_QUEUING).

IPAddressVersion **(MQCFIN)**
Specifies the IP version to be used (parameter identifier: MQIA_IP_ADDRESS_VERSION).

The value can be:

MQIPADDR_IPV4
The IPv4 stack is used.

MQIPADDR_IPV6
The IPv6 stack is used.

ListenerTimer **(MQCFIN)**
The time interval, in seconds, between attempts to restart a listener following an APPC or TCP/IP failure (parameter identifier: MQCA_LISTENER_TIMER).

LocalEvent **(MQCFIN)**
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

LU62ARMSuffix **(MQCFST)**
The suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator (parameter identifier: MQCA_LU62_ARM_SUFFIX).

The maximum length of this name is MQ_ARM_SUFFIX_LENGTH.

LU62Channels **(MQCFIN)**
Maximum number of current channels that use the LU 6.2 transmission protocol, including clients connected to server connection channels (parameter identifier: MQIA_LU62_CHANNELS).

LUGroupName **(MQCFST)**
The generic LU name that the LU 6.2 listener that handles inbound transmissions for the queue sharing group is to use. This name must be the same as *LUName* (parameter identifier: MQCA_LU_GROUP_NAME).

The maximum length of this name is MQ_LU_NAME_LENGTH.

LUName **(MQCFST)**
The LU name that the LU 6.2 listener that handles outbound transmissions is to use. This name must be the same as *LUGroupName* (parameter identifier: MQCA_LU_NAME).

The maximum length of this name is MQ_LU_NAME_LENGTH.

MaxActiveChannels **(MQCFIN)**
Maximum number of channels that can be active at the same time (parameter identifier: MQIA_ACTIVE_CHANNELS).

MaxChannels **(MQCFIN)**
Maximum number of current channels, including clients connected to server connection channels (parameter identifier: MQIA_MAX_CHANNELS).

MaxHandles **(MQCFIN)**
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

Specifies the maximum number of handles that any one job can have open at the same time.

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

MaxPriority **(MQCFIN)**
Maximum priority (parameter identifier: MQIA_MAX_PRIORITY).

MaxUncommittedMsgs **(MQCFIN)**
Maximum number of uncommitted messages within a unit of work (parameter identifier: MQIA_MAX_UNCOMMITTED_MSGS).

That is:

- The number of messages that can be retrieved, plus
- The number of messages that can be put on a queue, plus
- Any trigger messages generated within this unit of work

under any one syncpoint. This limit does not apply to messages that are retrieved or put outside syncpoint.

OutboundPortMax **(MQCFIN)**
Outbound port range maximum (parameter identifier: MQIA_OUTBOUND_PORT_MAX).

The upper limit for the range of port numbers used when binding outgoing channels.

OutboundPortMin **(MQCFIN)**
Outbound port range minimum (parameter identifier: MQIA_OUTBOUND_PORT_MIN).

The lower limit for the range of port numbers used when binding outgoing channels.

PerformanceEvent **(MQCFIN)**
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

Platform **(MQCFIN)**
Platform on which the queue manager resides (parameter identifier: MQIA_PLATFORM).

QMgrDesc **(MQCFST)**
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

QMgrIdentifier **(MQCFST)**
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

QMgrName **(MQCFST)**
Name of local queue manager (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QSGName **(MQCFST)**
Queue sharing group name (parameter identifier: MQCA_QSG_NAME).

The maximum length of the string is MQ_QSG_NAME_LENGTH.

QueueAccounting **(MQCFIN)**
Specifies whether accounting information is collected for queues (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_ON
For all queues that have the queue parameter *QueueAccounting* specified as MQMON_Q_MGR, accounting information is collected.

MQMON_OFF
For all queues that have the queue parameter *QueueAccounting* specified as MQMON_Q_MGR, accounting information is not collected.

MQMON_NONE
Accounting information is not collected for queues.

QueueMonitoring **(MQCFIN)**
Level of real-time monitoring data collection for queues (parameter identifier: MQIA_MONITORING_Q).

The value can be:

MQMON_NONE
Monitoring data collection is disabled, regardless of the setting for the *QueueMonitoring* queue attribute.

MQMON_OFF
Monitoring data collection is turned off for queues specifying MQMON_Q_MGR in the *QueueMonitoring* queue attribute.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection for queues specifying MQMON_Q_MGR in the *QueueMonitoring* queue attribute.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection for queues specifying MQMON_Q_MGR in the *QueueMonitoring* queue attribute.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection for queues specifying MQMON_Q_MGR in the *QueueMonitoring* queue attribute.

ReceiveTimeout **(MQCFIN)**
In conjunction with *ReceiveTimeoutType* specifies how long a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: MQIA_RECEIVE_TIMEOUT).

ReceiveTimeoutMin **(MQCFIN)**
The minimum time, in seconds, that a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: MQIA_RECEIVE_TIMEOUT_MIN).

ReceiveTimeoutType **(MQCFIN)**
In conjunction with *ReceiveTimeout* specifies how long a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: MQIA_RECEIVE_TIMEOUT_TYPE).

The value can be:

MQRCVTIME_MULTIPLY
The *ReceiveTimeout* value is a multiplier to be applied to the negotiated value of *HeartbeatInterval* to determine how long a channel will wait. This is the queue manager's initial default value.

MQRCVTIME_ADD
ReceiveTimeout is a value, in seconds, to be added to the negotiated value of *HeartbeatInterval* to determine how long a channel will wait.

MQRCVTIME_EQUAL
ReceiveTimeout is a value, in seconds, representing how long a channel will wait.

RemoteEvent **(MQCFIN)**
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

RepositoryName **(MQCFST)**
Repository name (parameter identifier: MQCA_REPOSITORY_NAME).

The name of a cluster for which this queue manager is to provide a repository service.

RepositoryNamelist **(MQCFST)**
Repository name list (parameter identifier: MQCA_REPOSITORY_NAMELIST).

The name of a list of clusters for which this queue manager is to provide a repository service.

SharedQueueQueueManagerName **(MQCFIN)**
Specifies how messages are put on a shared queue that specifies another queue manager from a queue sharing group as the object queue manager (parameter identifier: MQIA_SHARED_Q_Q_MGR_NAME).

The value can be:

MQSQQM_USE
Messages are delivered to the object queue manager before being put on the shared queue.

MQSQQM_IGNORE
Messages are put directly on the shared queue.

SSLCRLNameList **(MQCFST)**
SSL CRL name list (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

SSLEvent **(MQCFIN)**
Determines whether IMS bridge events are generated (parameter identifier: MQIA_SSL_EVENT).

The value can be:

MQEVR_ENABLED
All SSL events are enabled.

MQEVR_DISABLED
All SSL events are disabled.

SSLKeyRepository **(MQCFST)**
SSL key repository (parameter identifier: MQCA_SSL_KEY_REPOSITORY).

The maximum length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLKeyResetCount **(MQCFIN)**
SSL key reset count (parameter identifier: MQIA_SSL_RESET_COUNT).

The maximum length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLTasks **(MQCFIN)**
SSL tasks (parameter identifier: MQIA_SSL_TASKS).

StartStopEvent **(MQCFIN)**
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

SyncPoint **(MQCFIN)**
Syncpoint availability (parameter identifier: MQIA_SYNCPOINT).

TCPChannels **(MQCFIN)**
Maximum number of current channels that use the TCP/IP transmission protocol, including clients connected to server connection channels (parameter identifier: MQIA_TCP_CHANNELS).

TCPKeepAlive **(MQCFIN)**
Specifies whether to use the TCP KEEPALIVE facility to check whether the MCA at the opposite end of a channel is available (parameter identifier: MQIA_TCP_KEEP_ALIVE).

The value can be:

MQTCPKEEP_YES
Use the TCP KEEPALIVE facility as specified in the TCP profile configuration data set.

MQTCPKEEP_NO
Do not use the TCP KEEPALIVE facility.

TCPName **(MQCFST)**
TCP name (parameter identifier: MQIA_TCP_NAME).

The name of the current TCP/IP system in use.

The maximum length of this value is MQ_TCP_NAME_LENGTH.

TCPStackType **(MQCFIN)**
TCP stack type (parameter identifier: MQIA_TCP_STACK_TYPE).

Specifies whether the channel initiator uses the TCP/IP address space specified in TCPNAME only, or whether it can bind to any selected TCP/IP address.

The value can be:

MQTCPSTACK_SINGLE
The channel initiator uses the TCP/IP address space specified in TCPNAME only.

MQTCPSTACK_MULTIPLE
The initiator can use any TCP/IP address space available to it. If no other address spaces are available, the address space specified in TCPNAME is used.

TraceRouteRecording **(MQCFIN)**
Specifies whether trace-route messaging is enabled or disabled (parameter identifier: MQIA_TRACE_ROUTE_RECORDING).

The value can be:

MQRECORDING_MSG
Trace-route messaging is enabled. Trace-route reply messages are delivered to the reply-to queue specified in the message descriptor of the message.

MQRECORDING_Q
Trace-route messaging is enabled. Trace-route reply messages are delivered to a fixed name queue.

MQRECORDING_DISABLED.
Trace-route messaging is disabled.

TriggerInterval **(MQCFIN)**
Trigger interval (parameter identifier: MQIA_TRIGGER_INTERVAL).

Specifies the trigger time interval, expressed in milliseconds, for use only with queues where *TriggerType* has a value of MQTT_FIRST.

Storage class attributes

Event messages relating to objects can include storage class attributes

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

PageSetId **(MQCFIN)**
Page set identifier (parameter identifier: MQIA_PAGESET_ID).

PassTicketApplication **(MQCFST)**
Name of the application used to authenticate IMS bridge passtickets (parameter identifier: MQCA_PASS_TICKET_APPL).

The maximum length of the string is MQ_PASS_TICKET_APPL_LENGTH.

StgClassDesc **(MQCFST)**
Storage class description (parameter identifier: MQCA_STORAGE_CLASS_DESC).

The maximum length of the string is MQ_STORAGE_CLASS_DESC_LENGTH.

XCFGroupName **(MQCFST)**
XCF group name (parameter identifier: MQCA_XCF_GROUP_NAME).

The maximum length of the string is MQ_XCF_GROUP_NAME_LENGTH.

XCFMemberName **(MQCFST)**
XCF member name (parameter identifier: MQCA_XCF_MEMBER_NAME).

The maximum length of the string is MQ_XCF_MEMBER_NAME_LENGTH.

Topic attributes

Event messages relating to objects can include topic attributes

AlterationDate **(MQCFST)**
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered, in the form `yyyy-mm-dd`.

AlterationTime **(MQCFST)**
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered, in the form `hh.mm.ss` .

ClusterName **(MQCFST)**
The name of the cluster to which this topic belongs (parameter identifier: MQCA_CLUSTER_NAME).

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

The value can be as follows:

Blank This topic does not belong to a cluster. Publications and subscriptions for this topic are not propagated to publish/subscribe cluster-connected queue managers.

This is the default value for this parameter if no value is specified.

String This topic belongs to the indicated cluster.

Additionally, if PublicationScope or SubscriptionScope is set to MQSCOPE_ALL, this cluster is to be used for the propagation of publications and subscriptions, for this topic, to publish/subcribe cluster-connected queue managers.

DefPersistence **(MQCFIN)**
Default persistence (parameter identifier: MQIA_TOPIC_DEF_PERSISTENCE).

The value can be:

MQPER_PERSISTENCE_AS_PARENT
The default persistence is based on the setting of the closest parent administrative topic object in the topic tree.

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

DefPriority **(MQCFIN)**
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

DefPutResponse **(MQCFIN)**
Default put response (parameter identifier: MQIA_DEF_PUT_RESPONSE_TYPE).

The value can be:

MQPRT_ASYNC_RESPONSE
The put operation is issued asynchronously, returning a subset of MQMD fields.

MQPRT_RESPONSE_AS_PARENT
The default put response is based on the setting of the closest parent administrative topic object in the topic tree.

MQPRT_SYNC_RESPONSE
The put operation is issued synchronously, returning a response.

DurableModelQName **(MQCFST)**
Name of the model queue to be used for durable managed subscriptions (parameter identifier: MQCA_MODEL_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

DurableSubscriptions **(MQCFIN)**
Whether applications are permitted to make durable subscriptions (parameter identifier: MQIA_DURABLE_SUB).

The value can be:

MQSUB_DURABLE_AS_PARENT
Whether durable subscriptions are permitted is based on the setting of the closest parent administrative topic object in the topic tree.

MQSUB_DURABLE
Durable subscriptions are permitted.

MQSUB_NON_DURABLE
Durable subscriptions are not permitted.

InhibitPublications **(MQCFIN)**
Whether publications are allowed for this topic (parameter identifier: MQIA_INHIBIT_PUB).

The value can be:

MQTA_PUB_AS_PARENT
Whether messages can be published to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_PUB_INHIBITED
Publications are inhibited for this topic.

MQTA_PUB_ALLOWED
Publications are allowed for this topic.

InhibitSubscriptions **(MQCFIN)**
Whether subscriptions are allowed for this topic (parameter identifier: MQIA_INHIBIT_SUB).

The value can be:

MQTA_SUB_AS_PARENT
Whether applications can subscribe to this topic is based on the setting of the closest parent administrative topic object in the topic tree.

MQTA_SUB_INHIBITED
Subscriptions are inhibited for this topic.

MQTA_SUB_ALLOWED
Subscriptions are allowed for this topic.

NonDurableModelQName **(MQCFST)**
Name of the model queue to be used for non durable managed subscriptions (parameter identifier: MQCA_MODEL_NON_DURABLE_Q).

The maximum length of the string is MQ_Q_NAME_LENGTH.

NonPersistentMsgDelivery **(MQCFIN)**
The delivery mechanism for non-persistent messages published to this topic (parameter identifier: MQIA_NPM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Non-persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Non-persistent messages must be delivered to all durable subscribers. Failure to deliver a non-persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Non-persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

PersistentMsgDelivery **(MQCFIN)**
The delivery mechanism for persistent messages published to this topic (parameter identifier: MQIA_PM_DELIVERY).

The value can be:

MQDLV_AS_PARENT
The delivery mechanism used is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

MQDLV_ALL
Persistent messages must be delivered to all subscribers, irrespective of durability for the MQPUT call to report success. If a delivery failure to any subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_DUR
Persistent messages must be delivered to all durable subscribers. Failure to deliver a persistent message to any non-durable subscribers does not return an error to the MQPUT call. If a delivery failure to a durable subscriber occurs, no other subscribers receive the message and the MQPUT fails.

MQDLV_ALL_AVAIL
Persistent messages are delivered to all subscribers that can accept the message. Failure to deliver the message to any subscriber does not prevent other subscribers from receiving the message.

ProxySubscriptions **(MQCFIN)**
Whether a proxy subscription is to be sent for this topic, even if no local subscriptions exist, to directly connected queue managers (parameter identifier: MQIA_PROXY_SUB).

The value can be:

MQTA_PROXY_SUB_FORCE
A proxy subscription is sent to connected queue managers even if no local subscriptions exist.

MQTA_PROXY_SUB_FIRSTUSE
A proxy subscription is sent for this topic only when a local subscription exists.

PublicationScope **(MQCFIN)**
Whether this queue manager propagates publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_PUB_SCOPE).

The value can be:

MQSCOPE_ALL
Publications for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

MQSCOPE_AS_PARENT
Whether this queue manager will propagate publications to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

This is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Publications for this topic are not propagated to other queue managers.

Note: You can override this behavior on a publication-by-publication basis, using MQPMO_SCOPE_QMGR on the Put Message Options.

QMgrName **(MQCFST)**
Name of local queue manager (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH

SubscriptionScope **(MQCFIN)**
Whether this queue manager propagates subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster (parameter identifier: MQIA_SUB_SCOPE).

The value can be:

MQSCOPE_ALL
Subscriptions for this topic are propagated to hierarchically connected queue managers and to publish/subscribe cluster-connected queue managers.

MQSCOPE_AS_PARENT
Whether this queue manager will propagate subscriptions to queue managers as part of a hierarchy or as part of a publish/subscribe cluster is based on the setting of the first parent administrative node found in the topic tree relating to this topic.

This is the default value for this parameter if no value is specified.

MQSCOPE_QMGR
Subscriptions for this topic are not propagated to other queue managers.

Note: You can override this behavior on a subscription-by-subscription basis, using MQSO_SCOPE_QMGR on the Subscription Descriptor or SUBSCOPE(QMGR) on DEFINE SUB.

TopicDesc **(MQCFST)**
Topic description (parameter identifier: MQCA_TOPIC_DESC).

The maximum length is MQ_TOPIC_DESC_LENGTH.

TopicName **(MQCFST)**
Topic object name (parameter identifier: MQCA_TOPIC_NAME).

The maximum length of the string is MQ_TOPIC_NAME_LENGTH

TopicString **(MQCFST)**
The topic string (parameter identifier: MQCA_TOPIC_STRING).

The '/' character within this string has special meaning. It delimits the elements in the topic tree. A topic string can start with the '/' character but is not required to. A string starting with the '/' character is not the same as the string which starts without the '/' character. A topic string cannot end with the "/" character.

The maximum length of the string is MQ_TOPIC_STR_LENGTH.

TopicType **(MQCFIN)**
Whether this object is a local or cluster topic (parameter identifier: MQIA_TOPIC_TYPE).

The value can be:

MQTOPT_LOCAL
This object is a local topic.

MQTOPT_CLUSTER
This object is a cluster topic.

WildcardOperation **(MQCFIN)**
Behavior of subscriptions including wildcards made to this topic (parameter identifier: MQIA_WILDCARD_OPERATION).

The value can be:

MQTA_PASSTHRU
Subscriptions made using wildcard topic names that are less specific than the topic string at this topic object will receive publications made to this topic and to topic strings more specific than this topic. This is the default supplied with WebSphere MQ.

MQTA_BLOCK
Subscriptions made using wildcard topic names that are less specific than the topic string at this topic object will not receive publications made to this topic or to topic strings more specific than this topic.

Event message reference

Use this page to obtain an overview of information about the format of event messages.

For each instrumentation event, information is returned in both the message descriptor and message data parts of the events messages.

Related concepts:

"Event message descriptions" on page 2915
The event message data contains information specific to the event that was generated. This data includes the name of the queue manager and, where appropriate, the name of the queue.

Related reference:

"Event message format" on page 2907
Event messages are standard WebSphere MQ messages containing a message descriptor and message data.

"Event message MQMD (message descriptor)" on page 2908
The message descriptor for an event message contains information that a system monitoring application can use, such as the message type and format, and the date and time that the message was put on the event queue.

“Event message MQCFH (PCF header)” on page 2913
The message data in event messages is in programmable command format (PCF), as used in PCF command inquiries and responses. The message data consists of two parts: the event header and the event data.

Related information:

Instrumentation events

Event message format

Event messages are standard WebSphere MQ messages containing a message descriptor and message data.

Table 264 shows the basic structure of event messages and, in the Event data column, the names of the fields in an event message for queue service interval events.

Table 264. Event message structure for queue service interval events

Message descriptor	**Message data**	
MQMD structure	**PCF header** **MQCFH structure**	**Event data** [1]
Structure identifier Structure version Report options Message type Expiration time Feedback code Encoding Coded character set ID Message format Message priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier Accounting token Application identity data Application type Application name Put date Put time Application origin data Group identifier Message sequence number Offset Message flags Original length	Structure type Structure length Structure version Command identifier Message sequence number Control options Completion code Reason code Parameter count	Queue manager name Queue name Time since last reset Maximum number of messages on queue Number of messages put to queue Number of messages retrieved from queue

Note:

1. The parameters shown are those returned for a queue service interval event. The actual event data depends on the specific event.

In general, you need only a subset of this information for any system management programs that you write. For example, your application might need the following data:

- The name of the application causing the event
- The name of the queue manager on which the event occurred
- The queue on which the event was generated
- The event statistics

Event message MQMD (message descriptor)

The message descriptor for an event message contains information that a system monitoring application can use, such as the message type and format, and the date and time that the message was put on the event queue.

The information in the descriptor informs a system management application that the message type is MQMT_DATAGRAM, and the message format is MQFMT_EVENT.

Many of the fields in an event message contain fixed data, which is supplied by the queue manager that generated the message. The MQMD also specifies the name of the queue manager (truncated to 28 characters) that put the message.

For an event message, the MQMD structure contains the following values:

StrucId

Description: Structure identifier.
Data type: MQCHAR4.
Value: MQMD_STRUC_ID

Version

Description: Structure version number.
Data type: MQLONG.
Values:

MQMD_VERSION_1
Version-1 message descriptor structure, supported in all environments.

MQMD_VERSION_2
Version-2 message descriptor structure, supported on AIX, HP-UX, z/OS, IBM i, Solaris, Linux, Windows, and all WebSphere MQ MQI clients connected to these systems.

Report

Description: Options for report messages.
Data type: MQLONG.
Value:

MQRO_NONE
No reports required.

MsgType

Description: Indicates type of message.
Data type: MQLONG.
Value: MQMT_DATAGRAM.

Expiry

Description: Message lifetime.
Data type: MQLONG.
Value:

MQEI_UNLIMITED
The message does not have an expiry time.

Feedback

Description: Feedback or reason code.
Data type: MQLONG.
Value: MQFB_NONE.

Encoding

Description: Numeric encoding of message data.
Data type: MQLONG.
Value: MQENC_NATIVE.

CodedCharSetId

Description: Character set identifier of event message data.
Data type: MQLONG.
Value: Coded character set ID (CCSID) of the queue manager generating the event.

Format

Description: Format name of message data.
Data type: MQCHAR8.
Value:

MQFMT_EVENT
Event message.

Priority

Description: Message priority.
Data type: MQLONG.
Value:

MQPRI_PRIORITY_AS_Q_DEF
The priority is that of the event queue.

Persistence

Description: Message persistence.
Data type: MQLONG.
Value:

MQPER_PERSISTENCE_AS_Q_DEF
The priority is that of the event queue.

MsgId

Description: Message identifier.
Data type: MQBYTE24.
Value: A unique value generated by the queue manager.

CorrelId

Description: Correlation identifier.
Data type: MQBYTE24.
Value:

For performance, queue manager, logger, channel, bridge, and SSL events:

MQCI_NONE
No correlation identifier is specified. This is for private queues only.

For such events on a shared queue, a nonzero correlation identifier is set. This parameter is set so that you can track multiple event messages from different queue managers. The characters are specified in the following way:

1–4 Product identifier ('CSQ ')

5–8 Queue-sharing group name

9 Queue manager identifier

10–17 Time stamp

18–24 Nulls

For configuration and command events:

A unique nonzero correlation identifier
All messages relating to the same event have the same CorrelId.

BackoutCount

Description: Backout counter.
Data type: MQLONG.
Value: 0.

ReplyToQ

Description: Name of reply queue.
Data type: MQCHAR48.
Values: Blank.

ReplyToQMgr

Description: Name of reply queue manager.
Data type: MQCHAR48.
Value: The queue manager name at the originating system.

UserIdentifier

Description: Identifies the application that originated the message.
Data type: MQCHAR12.
Value: Blank.

AccountingToken

Description: Accounting token that allows an application to charge for work done as a result of the message.
Data type: MQBYTE32.
Value: MQACT_NONE.

ApplIdentityData

Description: Application data relating to identity.
Data type: MQCHAR32.
Values: Blank.

PutApplType

Description: Type of application that put the message.
Data type: MQLONG.
Value:

MQAT_QMGR
Queue manager generated message.

PutApplName

Description: Name of application that put the message.
Data type: MQCHAR28.
Value: The queue manager name at the originating system.

PutDate

Description: Date when message was put.
Data type: MQCHAR8.
Value: As generated by the queue manager.

PutTime

Description: Time when message was put.
Data type: MQCHAR8.
Value: As generated by the queue manager.

ApplOriginData

Description: Application data relating to origin.
Data type: MQCHAR4.
Value: Blank.

Note: If *Version* is MQMD_VERSION_2, the following additional fields are present:

GroupId

Description: Identifies to which message group or logical message the physical message belongs.
Data type: MQBYTE24.
Value:

MQGI_NONE
No group identifier specified.

MsgSeqNumber

Description: Sequence number of logical message within group.
Data type: MQLONG.
Value: 1.

Offset

Description: Offset of data in physical message from start of logical message.
Data type: MQLONG.
Value: 0.

MsgFlags

Description: Message flags that specify attributes of the message or control its processing.
Data type: MQLONG.
Value: MQMF_NONE.

OriginalLength

Description: Length of original message.
Data type: MQLONG.
Value: MQOL_UNDEFINED.

Event message MQCFH (PCF header)

The message data in event messages is in programmable command format (PCF), as used in PCF command inquiries and responses. The message data consists of two parts: the event header and the event data.

The MQCFH header specifies the following information:

- The category of event: whether the event is a queue manager, performance, channel, configuration, command, or logger event.
- A reason code specifying the cause of the event. For events caused by MQI calls, this reason code is the same as the reason code for the MQI call.

Reason codes have names that begin with the characters MQRC_. For example, the reason code MQRC_PUT_INHIBITED is generated when an application attempts to put a message on a queue that is not enabled for puts.

For an event, the MQCFH structure contains the following values:

Type

Description: Structure type that identifies the content of the message.
Data type: MQLONG.
Value:

MQCFT_EVENT
Message is reporting an event.

StrucLength

Description: Structure length.
Data type: MQLONG.
Value:

MQCFH_STRUC_LENGTH
Length in bytes of MQCFH structure.

Version

Description: Structure version number.
Data type: MQLONG.
Values:

MQCFH_VERSION_1
Version-1 in all events except configuration and command events.

MQCFH_VERSION_2
Version-2 for configuration events.

MQCFH_VERSION_3
Version-3 for command events.

Command

Description: Command identifier. This identifies the event category.
Data type: MQLONG.
Values:

MQCMD_Q_MGR_EVENT
Queue manager event.

MQCMD_PERFM_EVENT
Performance event.

MQCMD_CHANNEL_EVENT
Channel event.

MQCMD_CONFIG_EVENT
Configuration event.

MQCMD_COMMAND_EVENT
Command event.

MQCMD_LOGGER_EVENT
Logger event.

MsgSeqNumber

Description: Message sequence number. This is the sequence number of the message within a group of related messages.
Data type: MQLONG.
Values:

1 For change object configuration events with attribute values before the changes, and for all other types of events.

2 For change object configuration events with the attribute values after the changes

Control

Description: Control options.
Data type: MQLONG.
Values:

MQCFC_LAST
For change object configuration events with attribute values after the changes, and for all other types of events.

MQCFC_NOT_LAST
For Change Object configurations events only, with the attribute values from before the changes.

CompCode

Description: Completion code.
Data type: MQLONG.
Values:

MQCC_OK
Event reporting OK condition.

MQCC_WARNING
Event reporting warning condition. All events have this completion code, unless otherwise specified.

Reason

Description:	Reason code qualifying completion code.
Data type:	MQLONG.
Values:	MQRC_* Dependent on the event being reported. **Note:** Events with the same reason code are further identified by the *ReasonQualifier* parameter in the event data.

ParameterCount

Description:	Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Data type:	MQLONG.
Values:	0 or greater.

Event message descriptions

The event message data contains information specific to the event that was generated. This data includes the name of the queue manager and, where appropriate, the name of the queue.

The data structures returned depend on which particular event was generated. In addition, for some events, certain parameters of the structures are optional, and are returned only if they contain information that is relevant to the circumstances giving rise to the event. The values in the data structures depend on the circumstances that caused the event to be generated.

Note:

1. The PCF structures in the message data are not returned in a defined order. They must be identified from the parameter identifiers shown in the description.
2. Events are available on all platforms, unless specific limitations are shown at the start of an event description.

Alias Base Queue Type Error:

Event name:	Alias Base Queue Type Error.
Reason code in MQCFH:	MQRC_ALIAS_BASE_Q_TYPE_ERROR (2001, X'7D1'). Alias base queue not a valid type.
Event description:	An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseObjectName* in the alias queue definition resolves to a queue that is not a local queue, or local definition of a remote queue.
Event type:	Local.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

BaseObjectName

Description: Object name to which the alias resolves.
Identifier: MQCA_BASE_OBJECT_NAME. For compatibility with existing applications you can still use MQCA_BASE_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

QType

Description: Type of queue to which the alias resolves.
Identifier: MQIA_Q_TYPE.
Data type: MQCFIN.
Values:

MQQT_ALIAS
Alias queue definition.

MQQT_MODEL
Model queue definition.

Returned: Always.

ApplType

Description: Type of the application making the call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Bridge Started:

Event name:	Bridge Started.
Reason code in MQCFH:	MQRC_BRIDGE_STARTED (2125, X'84D'). Bridge started.
Event description:	The IMS bridge has been started.
Event type:	IMS Bridge.
Platforms:	WebSphere MQ for z/OS only.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

BridgeType

Description: Bridge type.
Identifier: MQIACF_BRIDGE_TYPE.
Data type: MQCFIN.
Values:

MQBT_OTMA
OTMA bridge.

Returned: Always.

BridgeName

Description: Bridge name. For bridges of type MQBT_OTMA, the name is of the form `XCFgroupXCFmember`, where `XCFgroup` is the XCF group name to which both IMS and WebSphere MQ belong. `XCFmember` is the XCF member name of the IMS system.
Identifier: MQCACF_BRIDGE_NAME.
Data type: MQCFST.
Maximum length: MQ_BRIDGE_NAME_LENGTH.
Returned: Always.

Bridge Stopped:

Event name:	Bridge Stopped.
Reason code in MQCFH:	MQRC_BRIDGE_STOPPED (2126, X'84E'). Bridge stopped.
Event description:	The IMS bridge has been stopped.
Event type:	IMS Bridge.
Platforms:	WebSphere MQ for z/OS only.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code in MQCFH.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_BRIDGE_STOPPED_OK
Bridge has been stopped with either a zero return code or a warning return code. For MQBT_OTMA bridges, one side or the other issued a normal IXCLEAVE request.

MQRQ_BRIDGE_STOPPED_ERROR
Bridge has been stopped but there is an error reported.

Returned: Always.

BridgeType

Description:	Bridge type.
Identifier:	MQIACF_BRIDGE_TYPE.
Data type:	MQCFIN.
Value:	**MQBT_OTMA** OTMA bridge.
Returned:	Always.

BridgeName

Description:	Bridge name. For bridges of type MQBT_OTMA, the name is of the form `XCFgroupXCFmember`, where `XCFgroup` is the XCF group name to which both IMS and WebSphere MQ belong. `XCFmember` is the XCF member name of the IMS system.
Identifier:	MQCACF_BRIDGE_NAME.
Data type:	MQCFST.
Maximum length:	MQ_BRIDGE_NAME_LENGTH.
Returned:	Always.

ErrorIdentifier

Description:	When a bridge is stopped because of an error, this code identifies the error. If the event reports a bridge stop failure, the IMS sense code is set.
Identifier:	MQIACF_ERROR_IDENTIFIER.
Data type:	MQCFIN.
Returned:	If *ReasonQualifier* is MQRQ_BRIDGE_STOPPED_ERROR.

Change object:

Event name:	Change object.
Reason code in MQCFH:	MQRC_CONFIG_CHANGE_OBJECT (2368, X'940'). Existing object changed.
Event description:	An ALTER or DEFINE REPLACE command or an MQSET call was issued that successfully changed an existing object.
Event type:	Configuration.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Note: Two event messages are generated for the change object event. The first has the object attribute values **before** the change, the second has the attribute values **after** the change.

Event data

EventUserId

Description: The user id that issued the command or call that generated the event. (This is the same user id that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Datatype: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MQSET
MQSET call.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Datatype: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Datatype: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Datatype: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Datatype: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Datatype: MQCFIN.

Values:

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication record.

MQOT_NAMELIST
Namelist.

MQOT_NONE
No object.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

MQOT_COMM_INFO
Communication information.

MQOT_LISTENER
Channel Listener.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_Q_MGR_NAME
- MQCA_STORAGE_CLASS
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_NAME
- MQCA_COMM_INFO_NAME
- MQCACH_LISTENER_NAME

Note: MQCACH_CHANNEL_NAME can also be used for channel authentication.

Datatype: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.
Returned: Always

Disposition

Description:	Object disposition:
Identifier:	MQIA_QSG_DISP.
Datatype:	MQCFIN.
Values:	**MQQSGD_Q_MGR** Object resides on page set of queue manager. **MQQSGD_SHARED** Object resides in shared repository and messages are shared in coupling facility. **MQQSGD_GROUP** Object resides in shared repository. **MQQSGD_COPY** Object resides on page set of queue manager and is a local copy of a GROUP object.
Returned:	Always, except for queue manager and CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 2872.

Channel Activated:

Event name:	Channel Activated.
Reason code in MQCFH:	MQRC_CHANNEL_ACTIVATED (2295, X'8F7'). Channel activated.
Event description:	This condition is detected when a channel that has been waiting to become active, and for which a Channel Not Activated event has been generated, is now able to become active, because an active slot has been released by another channel. This event is not generated for a channel that is able to become active without waiting for an active slot to be released.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description: Channel Name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Only for commands that do not contain a generic name.

Channel Auto-definition Error:

Event name:	Channel Auto-definition Error.
Reason code in MQCFH:	MQRC_CHANNEL_AUTO_DEF_ERROR (2234, X'8BA'). Automatic channel definition failed.
Event description:	This condition is detected when the automatic definition of a channel fails; this may be because an error occurred during the definition process, or because the channel automatic-definition exit inhibited the definition. Additional information indicating the reason for the failure is returned in the event message.
Event type:	Channel.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ChannelName

Description: Name of the channel for which the auto-definiton has failed.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ChannelType

Description: Channel Type. This specifies the type of channel for which the auto-definition has failed.
Identifier: MQIACH_CHANNEL_TYPE.
Data type: MQCFIN.
Values:

MQCHT_RECEIVER
Receiver.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLUSSDR
Cluster-sender.

Returned: Always.

ErrorIdentifier

Description: Identifier of the cause of the error. This contains either the reason code (MQRC_* or MQRCCF_*) resulting from the channel definition attempt or the value MQRCCF_SUPPRESSED_BY_EXIT if the attempt to create the definition was disallowed by the exit.
Identifier: MQIACF_ERROR_IDENTIFIER.
Data type: MQCFIN.
Returned: Always.

ConnectionName

Description: Name of the partner attempting to establish connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always.

AuxErrorDataInt1

Description: Auxiliary error data. This contains the value returned by the exit in the *Feedback* field of the MQCXP to indicate why the auto definition has been disallowed.
Identifier: MQIACF_AUX_ERROR_DATA_INT_1.
Data type: MQCFIN.
Returned: Only if *ErrorIdentifier* contains MQRCCF_SUPPRESSED_BY_EXIT.

Channel Auto-definition OK:

Event name:	Channel Auto-definition OK.
Reason code in MQCFH:	MQRC_CHANNEL_AUTO_DEF_OK (2233, X'8B9'). Automatic channel definition succeeded.
Event description:	This condition is detected when the automatic definition of a channel is successful. The channel is defined by the MCA.
Event type:	Channel.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ChannelName

Description: Name of the channel being defined.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ChannelType

Description: Type of channel being defined.
Identifier: MQIACH_CHANNEL_TYPE.
Data type: MQCFIN.
Values:

MQCHT_RECEIVER
Receiver.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLUSSDR
Cluster-sender.

Returned: Always.

ConnectionName

Description: Name of the partner attempting to establish connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always.

Channel Blocked:

Event name:	Channel Blocked.
Reason code in MQCFH:	MQRC_CHANNEL_BLOCKED Channel blocked. MQRC_CHANNEL_BLOCKED_WARNING Channel blocked – warning mode.
Event description:	This event is issued when an attempt to start an inbound channel is blocked. For MQRC_CHANNEL_BLOCKED_WARNING, temporary access has been granted to the channel because the channel authentication record is defined with WARN set to YES.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

Reason qualifier

Description: Identifier that qualifies the reason code
Identifier: MQIACF_REASON_QUALIFIER
Data type: MQCFIN.
Values:

MQRQ_CHANNEL_BLOCKED_ADDRESS
Channel was blocked due to its IP address being in the list to be refused

MQRQ_CHANNEL_BLOCKED_USERID
Channel was blocked due to its asserted or mapped user ID being in the list to be refused.

MQRQ_CHANNEL_BLOCKED_NOACCESS
Channel was blocked due to its IP address; SSL Peer name; remote queue manager name or client user ID being mapped to have no access.

Returned: Always.

ChannelName

Description: Channel Name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: If the Reason Qualifier is not MQRQ_CHANNEL_BLOCKED_ADDRESS. In that case the inbound connection is blocked before the channel name is known.

UserIdentifier

Description: User identifier that was blocked.
Identifier: MQCACF_USER_IDENTIFIER
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH
Returned: Only if the Reason Qualifier is MQRQ_CHANNEL_BLOCKED_USERID

ConnectionName

Description: Address of the partner attempting to establish connection
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always

RemoteQMgrName

Description: Name of the partner queue manager attempting to establish connection.
Identifier: MQCA_REMOTE_Q_MGR_NAME
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH
Returned: Only for inbound queue manager connections.

SSLPeerName

Description: The Distinguished Name in the certificate sent from the remote system.
Identifier: MQCACH_SSL_PEER_NAME
Data type: MQCFST.
Maximum length: MQ_DISTINGUISHED_NAME_LENGTH
Returned: Whenever the channel is using SSL and the client has not connected anonymously.

ClientUserIdentifier

Description: Client side user identifier of the partner attempting to establish connection.
Identifier: MQCACH_CLIENT_USER_ID
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH
Returned: Only for inbound client connections, if the Reason Qualifier is not MQRQ_CHANNEL_BLOCKED_ADDRESS. In that case the inbound connection is blocked before the client user Id name is known.

ApplType

Description:	Type of application that made the API call.
Identifier:	MQIA_APPL_TYPE
Data type:	MQCFIN.
Returned:	Only for inbound client connections. If the Reason Qualifier is not MQRQ_CHANNEL_BLOCKED_ADDRESS. In that case the inbound connection is blocked before the application name is known.

ApplName

Description:	Name of the application that made the API call.
Identifier:	MQCACF_APPL_NAME
Data type:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH
Returned:	Only for inbound client connections. If the Reason Qualifier is not MQRQ_CHANNEL_BLOCKED_ADDRESS. In that case the inbound connection is blocked before the application name is known.

Channel Conversion Error:

Event name:	Channel Conversion Error.
Reason code in MQCFH:	MQRC_CHANNEL_CONV_ERROR (2284, X'8EC'). Channel conversion error.
Event description:	This condition is detected when a channel is unable to carry out data conversion and the MQGET call to get a message from the transmission queue resulted in a data conversion error. The reason for the failure is identified by *ConversionReasonCode*.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ConversionReasonCode

Description: Identifier of the cause of the conversion error.
Identifier: MQIACF_CONV_REASON_CODE.
Data type: MQCFIN.
Values:

MQRC_CONVERTED_MSG_TOO_BIG (2120, X'848')
Converted message too big for application buffer.

MQRC_FORMAT_ERROR (2110, X'83E')
Message format not valid.

MQRC_NOT_CONVERTED (2119, X'847')
Application message data not converted.

MQRC_SOURCE_CCSID_ERROR (2111, X'83F')
Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR (2113, X'841')
Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR (2114, X'842')
Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR (2112, X'840')
Integer encoding in message not recognized.

MQRC_TARGET_CCSID_ERROR (2115, X'843')
Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR (2117, X'845')
Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR (2118, X'846')
Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR (2116, X'844')
Integer encoding specified by receiver not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED (2079, X'81F')
Truncated message returned (processing completed).

MQRC_TRUNCATED_MSG_FAILED (2080, X'820')
Truncated message returned (processing not completed).

Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

Format

Description:	Format name.
Identifier:	MQCACH_FORMAT_NAME.
Data type:	MQCFST.
Maximum length:	MQ_FORMAT_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCACH_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Always.

Channel Not Activated:

Event name:	Channel Not Activated.
Reason code in MQCFH:	MQRC_CHANNEL_NOT_ACTIVATED (2296, X'8F8'). Channel cannot be activated.
Event description:	This condition is detected when a channel is required to become active, either because it is starting, or because it is about to make another attempt to establish connection with its partner. However, it is unable to do so because the limit on the number of active channels has been reached. See the following: • MaxActiveChannels parameter in the qm.ini file for AIX, HP-UX, and Solaris • MaxActiveChannels parameter in the Registry for Windows. • ACTCHL parameter on the ALTER QMGR command for z/OS The channel waits until it is able to take over an active slot released when another channel ceases to be active. At that time a Channel Activated event is generated.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: For sender, server, cluster-sender, and cluster-receiver channel types only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Only for commands that do not contain a generic name.

Channel Not Available:

Event name:	Channel Not Available.
Reason code in MQCFH:	MQRC_CHANNEL_NOT_AVAILABLE (2537, X'9E9'). Channel not available.
Event description:	This is issued when an attempt to start an inbound channel is rejected.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_MAX_ACTIVE_CHANNELS
Channel was unavailable due to maximum active channel instances (MaxActiveChannels qm.ini stanza on distributed or ACTCHL MQSC keyword on z/OS) limit being reached for the queue manager.

MQRQ_MAX_CHANNELS
Channel was unavailable due to maximum channel instances (MaxChannels qm.ini stanza on distributed or MAXCHL MQSC keyword on z/OS) limit being reached for the queue manager.

MQRQ_SVRCONN_INST_LIMIT
Channel was unavailable due to maximum active channel instances (MAXINST) limit being reached for the channel.

MQRQ_CLIENT_INST_LIMIT
Channel was unavailable due to maximum active channel instances (MAXINSTC) limit being reached for the client for the channel.

MQRQ_CAF_NOT_INSTALLED (z/OS only)
Channel was unavailable due to the client attach feature not being installed.

Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ConnectionName

Description: Address of the partner attempting to establish connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always.

MaximumActiveChannels

Description: Maximum active channels.
Identifier: MQIA_ACTIVE_CHANNELS
Data type: MQCFIN.
Returned: Only where reason qualifier MQRQ_MAX_ACTIVE_CHANNELS.

MaximumChannels

Description: Maximum channels.
Identifier: MQIA_MAX_CHANNELS
Data type: MQCFIN
Returned: Only where reason qualifier MQRQ_MAX_CHANNELS.

MaximumInstances

Description: Maximum channel instances.
Identifier: MQIACH_MAX_INSTANCES
Data type: MQCFIN
Returned: Only where reason qualifier MQRQ_SVRCONN_INST_LIMIT.

MaximumClientInstances

Description: Maximum channel instances per client.
Identifier: MQIACH_MAX_INSTS_PER_CLIENT
Data type: MQCFIN
Returned: Only where reason qualifier MQRQ_CLIENT_INST_LIMIT.

Channel SSL Error:

Event name:	Channel SSL Error.
Reason code in MQCFH:	MQRC_CHANNEL_SSL_ERROR (2371, X'943'). Channel SSL Error.
Event description:	This condition is detected when a channel using Secure Sockets Layer (SSL) or Transport Layer Security (TLS) fails to establish a connection. *ReasonQualifier* identifies the nature of the error.
Event type:	SSL.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_SSL_HANDSHAKE_ERROR
The key exchange / authentication failure arose during the SSL or TLS handshake.

MQRQ_SSL_CIPHER_SPEC_ERROR
This error can mean any one of the following:

- The SSL or TLS client CipherSpec does not match that on the SSL or TLS server channel definition.
- An invalid CipherSpec has been specified.
- A CipherSpec has only been specified on one end of the SSL or TLS channel.

MQRQ_SSL_PEER_NAME_ERROR
The Distinguished Name in the certificate sent by one end of the SSL or TLS channel does not match the peer name on the end of the channel definition at the other end of the SSL or TLS channel.

MQRQ_SSL_CLIENT_AUTH_ERROR
The SSL or TLS server channel definition specified either SSLCAUTH(REQUIRED) or a SSLPEER value that was not blank, but the SSL or TLS client did not provide a certificate.

Returned: Always.

ChannelName

Description: Channel Name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: The *ChannelName* might not be available if the channel has not yet got far enough through its start-up process, in this case the channel name will not be returned. Otherwise always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Returned: For sender, server, cluster-sender and cluster-receiver channels only.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the ConnectionName field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	The *ConnectionName* might not be available if the channel has not yet got far enough through its start-up process, in this case the connection name will not be returned. Otherwise always.

SSLHandshakeStage

Description:	Information about the SSL or TLS function call giving the error. For z/OS, details of function names can be found in the *System Secure Sockets Layer Programming Guide and Reference* SC24-5877.
Identifier:	MQCACH_SSL_HANDSHAKE_STAGE.
Data type:	MQCFST.
Maximum length:	MQ_SSL_HANDSHAKE_STAGE_LENGTH.
Returned:	This field is only present if *ReasonQualifier* is set to MQRQ_SSL_HANDSHAKE_ERROR.

SSLReturnCode

Description:	A numeric return code from a failing SSL or TLS call. Details of SSL or TLS Return Codes for specific platforms can be found as follows: • For platforms other than z/OS, see Secure Sockets Layer (SSL) return codes.
Identifier:	MQIACH_SSL_RETURN_CODE.
Data type:	MQCFIN.
Returned:	This field is only present if *ReasonQualifier* is set to MQRQ_SSL_HANDSHAKE_ERROR.

SSLPeerName

Description:	The Distinguished Name in the certificate sent from the remote system.
Identifier:	MQCACH_SSL_PEER_NAME.
Data type:	MQCFST.
Maximum length:	MQ_DISTINGUISHED_NAME_LENGTH.
Returned:	This field is only present if *ReasonQualifier* is set to MQRQ_SSL_PEER_NAME_ERROR and is not always present for this reason qualifier.

Channel SSL Warning:

Event name:	Channel SSL Warning.
Reason code in MQCFH:	MQRC_CHANNEL_SSL_WARNING (2552, X'9F8'). Channel SSL Warning.
Event description:	This condition is detected when a channel using Secure Sockets Layer (SSL) or Transport Layer Security (TLS) experiences a problem that does not cause it to fail to establish an SSL or TLS connection. *ReasonQualifier* identifies the nature of the event.
Event type:	SSL.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_SSL_UNKNOWN_REVOCATION
An OCSP responder returned a response of Unknown. WebSphere MQ is configured to produce warnings but allow the connection to continue.

Returned: Always.

ChannelName

Description: Channel Name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: The *ChannelName* might not be available if the channel has not yet got far enough through its start-up process, in this case the channel name will not be returned. Otherwise always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Returned: For sender, server, cluster-sender and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the ConnectionName field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: The *ConnectionName* may not be available if the channel has not yet got far enough through its start-up process, in this case the connection name will not be returned. Otherwise always.

Channel Started:

Event name:	Channel Started.
Reason code in MQCFH:	MQRC_CHANNEL_STARTED (2282, X'8EA'). Channel started.
Event description:	Either an operator has issued a Start Channel command, or an instance of a channel has been successfully established. This condition is detected when Initial Data negotiation is complete and resynchronization has been performed where necessary, such that message transfer can proceed.
Event type:	Channel.
Platforms:	All. Client connections do not produce this event.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description:	Channel name.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCACH_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Only for commands that do not contain a generic name.

Channel Stopped:

Event name:	Channel Stopped.
Reason code in MQCFH:	MQRC_CHANNEL_STOPPED (2283, X'8EB'). Channel stopped.
Event description:	This is issued when a channel instance stops. It will only be issued if the channel instance previously issued a channel started event.
Event type:	Channel.
Platforms:	All. Client connections do not produce this event.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_CHANNEL_STOPPED_OK
Channel has been closed with either a zero return code or a warning return code.

MQRQ_CHANNEL_STOPPED_ERROR
Channel has been closed but there is an error reported and the channel is not in stopped or retry state.

MQRQ_CHANNEL_STOPPED_RETRY
Channel has been closed and it is in retry state.

MQRQ_CHANNEL_STOPPED_DISABLED
Channel has been closed and it is in a stopped state.

Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ErrorIdentifier

Description: Identifier of the cause of the error. If a channel is stopped due to an error, this is the code that identifies the error. If the event message is because of a channel stop failure, the following fields are set:

1. *ReasonQualifier*, containing the value MQRQ_CHANNEL_STOPPED_ERROR
2. *ErrorIdentifier*, containing the code number of an error message that describes the error
3. *AuxErrorDataInt1*, containing error message integer insert 1
4. *AuxErrorDataInt2*, containing error message integer insert 2
5. *AuxErrorDataStr1*, containing error message string insert 1
6. *AuxErrorDataStr2*, containing error message string insert 2
7. *AuxErrorDataStr3*, containing error message string insert 3

The meanings of the error message inserts depend on the code number of the error message. Details of error-message code numbers and the inserts for specific platforms can be found as follows:

- For platforms other than z/OS, the last four digits of *ErrorIdentifier* when displayed in hexadecimal notation indicate the decimal code number of the error message.

  For example, if *ErrorIdentifier* has the value X'xxxxyyyy', the message code of the error message explaining the error is AMQyyyy. See “Diagnostic messages: AMQ4000-9999” on page 3026 for a description of these error messages.

Identifier: MQIACF_ERROR_IDENTIFIER.
Data type: MQCFIN.
Returned: Always.

AuxErrorDataInt1

Description: First integer of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the first integer parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQIACF_AUX_ERROR_DATA_INT_1.
Data type: MQCFIN.
Returned: Always.

AuxErrorDataInt2

Description: Second integer of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the second integer parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQIACF_AUX_ERROR_DATA_INT_2.
Data type: MQCFIN.
Returned: Always.

AuxErrorDataStr1

Description: First string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the first string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_1.
Data type: MQCFST.
Returned: Always.

AuxErrorDataStr2

Description: Second string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the second string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_2.
Data type: MQCFST.
Returned: Always.

AuxErrorDataStr3

Description: Third string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the third string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_3.
Data type: MQCFST.
Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Only for commands that do not contain a generic name.

Channel Stopped By User:

Event name:	Channel Stopped By User.
Reason code in MQCFH:	MQRC_CHANNEL_STOPPED_BY_USER (2279, X'8E7'). Channel stopped by user.
Event description:	This is issued when a user issues a STOP CHL command. *ReasonQualifier* identifies the reasons for stopping.
Event type:	Channel.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier that qualifies the reason code.
Identifier:	MQIACF_REASON_QUALIFIER.
Data type:	MQCFIN.
Values:	**MQRQ_CHANNEL_STOPPED_DISABLED** Channel has been closed and it is in a stopped state.
Returned:	Always.

ChannelName

Description:	Channel name.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

Command:

Event name:	Command.
Reason code in MQCFH:	MQRC_COMMAND_MQSC (2412, X'96C'). MQSC command successfully issued, or, MQRC_COMMAND_PCF (2413, X'96D'). PCF command successfully issued.
Event description:	Command successfully issued.
Event type:	Command.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.COMMAND.EVENT.

Event data

The event data consists of two groups, *CommandContext* and *CommandData*.

CommandContext

Description: PCF group containing the elements related to the context of the issued command.
Identifier: MQGACF_COMMAND_CONTEXT.
Data type: MQCFGR.
PCF elements in group:
- *EventUserId*
- *EventOrigin*
- *EventQMgr*
- *EventAccountingToken*
- *EventIdentityData*
- *EventApplType*
- *EventApplName*
- *EventApplOrigin*
- *Command*

Returned: Always.

EventUserId

Description: The user ID that issued the command or call that generated the event. (This is the same user ID that is used to check the authority to issue the command; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventIdentityData

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Data type: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Data type: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Data type: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

Command

Description: The command code.
Identifier: MQIACF_COMMAND.
Data type: MQCFIN.
Values:

- If the event relates to a PCF command, then the value is that of the Command parameter in the MQCFH structure in the command message.
- If the event relates to an MQSC command, then the value is as follows:

MQCMD_ARCHIVE_LOG
ARCHIVE LOG

MQCMD_BACKUP_CF_STRUC
BACKUP CFSTRUCT

MQCMD_CHANGE_AUTH_INFO
ALTER AUTHINFO

MQCMD_CHANGE_BUFFER_POOL
ALTER BUFFPOOL

MQCMD_CHANGE_CF_STRUC
ALTER CFSTRUCT

MQCMD_CHANGE_CHANNEL
ALTER CHANNEL

MQCMD_CHANGE_COMM_INFO
ALTER COMMINFO

MQCMD_CHANGE_LISTENER
ALTER LISTENER

MQCMD_CHANGE_NAMELIST
ALTER NAMELIST

MQCMD_CHANGE_PAGE_SET
ALTER PSID

MQCMD_CHANGE_PROCESS
ALTER PROCESS

MQCMD_CHANGE_Q
ALTER QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_CHANGE_Q_MGR
ALTER QMGR, DEFINE MAXSMSGS

MQCMD_CHANGE_SECURITY
ALTER SECURITY

MQCMD_CHANGE_SERVICE
ALTER SERVICE

MQCMD_CHANGE_STG_CLASS
ALTER STGCLASS

MQCMD_CHANGE_SUBSCRIPTION
ALTER SUBSCRIPTION

MQCMD_CHANGE_TOPIC
ALTER TOPIC

MQCMD_CHANGE_TRACE
ALTER TRACE

MQCMD_CLEAR_Q
CLEAR QLOCAL

MQCMD_CLEAR_TOPIC_STRING
CLEAR TOPICSTR

MQCMD_CREATE_AUTH_INFO
DEFINE AUTHINFO

MQCMD_CREATE_BUFFER_POOL
DEFINE BUFFPOOL

MQCMD_CREATE_CF_STRUC
DEFINE CFSTRUCT

MQCMD_CREATE_CHANNEL
DEFINE CHANNEL

MQCMD_CREATE_COMM_INFO
DEFINE COMMINFO

MQCMD_CREATE_LISTENER
DEFINE LISTENER

MQCMD_CREATE_NAMELIST
DEFINE NAMELIST

MQCMD_CREATE_PAGE_SET
DEFINE PSID

MQCMD_CREATE_PROCESS
DEFINE PROCESS

MQCMD_CREATE_Q
DEFINE QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_CREATE_SERVICE
DEFINE SERVICE

MQCMD_CREATE_STG_CLASS
DEFINE STGCLASS

MQCMD_CREATE_SUBSCRIPTION
DEFINE SUB

MQCMD_CREATE_TOPIC
DEFINE TOPIC

MQCMD_DELETE_AUTH_INFO
DELETE AUTHINFO

MQCMD_DELETE_CF_STRUC
DELETE CFSTRUCT

MQCMD_DELETE_CHANNEL
DELETE CHANNEL

MQCMD_DELETE_COMM_INFO
DELETE COMMINFO

MQCMD_DELETE_LISTENER
DELETE LISTENER

MQCMD_DELETE_NAMELIST
DELETE NAMELIST

MQCMD_DELETE_PAGE_SET
DELETE PSID

MQCMD_DELETE_PROCESS
DELETE PROCESS

MQCMD_DELETE_Q
DELETE QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_DELETE_SERVICE
DELETE SERVICE

MQCMD_DELETE_STG_CLASS
DELETE STGCLASS

MQCMD_DELETE_SUBSCRIPTION
DELETE SUBSCRIPTION

MQCMD_DELETE_TOPIC
DELETE TOPIC

MQCMD_INQUIRE_ARCHIVE
DISPLAY ARCHIVE

MQCMD_INQUIRE_AUTH_INFO
DISPLAY AUTHINFO

MQCMD_INQUIRE_CF_STRUC
DISPLAY CFSTRUCT

MQCMD_INQUIRE_CF_STRUC_STATUS
DISPLAY CFSTATUS

MQCMD_INQUIRE_CHANNEL
DISPLAY CHANNEL

MQCMD_INQUIRE_CHANNEL_INIT
DISPLAY CHINIT

MQCMD_INQUIRE_CHANNEL_STATUS
DISPLAY CHSTATUS

MQCMD_INQUIRE_CHLAUTH_RECS
DISPLAY CHLAUTH

MQCMD_INQUIRE_CLUSTER_Q_MGR
DISPLAY CLUSQMGR

MQCMD_INQUIRE_CMD_SERVER
DISPLAY CMDSERV

MQCMD_INQUIRE_COMM_INFO
DISPLAY COMMINFO

MQCMD_INQUIRE_CONNECTION
DISPLAY CONN

MQCMD_INQUIRE_LISTENER
DISPLAY LISTENER

MQCMD_INQUIRE_LOG
DISPLAY LOG

MQCMD_INQUIRE_NAMELIST
DISPLAY NAMELIST

MQCMD_INQUIRE_PROCESS
DISPLAY PROCESS

MQCMD_INQUIRE_PUBSUB_STATUS
DISPLAY PUBSUB

MQCMD_INQUIRE_Q
DISPLAY QUEUE

MQCMD_INQUIRE_Q_MGR
DISPLAY QMGR, DISPLAY MAXSMSGS

MQCMD_INQUIRE_QSG
DISPLAY GROUP

MQCMD_INQUIRE_Q_STATUS
DISPLAY QSTATUS

MQCMD_INQUIRE_SECURITY
DISPLAY SECURITY

MQCMD_INQUIRE_SERVICE
DISPLAY SERVICE

MQCMD_INQUIRE_STG_CLASS
DISPLAY STGCLASS

MQCMD_INQUIRE_SUBSCRIPTION
DISPLAY SUB

MQCMD_INQUIRE_SUB_STATUS
DISPLAY SBSTATUS

MQCMD_INQUIRE_SYSTEM
DISPLAY SYSTEM

MQCMD_INQUIRE_THREAD
DISPLAY THREAD

MQCMD_INQUIRE_TOPIC
DISPLAY TOPIC

MQCMD_INQUIRE_TOPIC_STATUS
DISPLAY TPSTATUS

MQCMD_INQUIRE_TRACE
DISPLAY TRACE

MQCMD_INQUIRE_USAGE
DISPLAY USAGE

MQCMD_MOVE_Q
MOVE QLOCAL

MQCMD_PING_CHANNEL
PING CHANNEL

MQCMD_RECOVER_BSDS
RECOVER BSDS

MQCMD_RECOVER_CF_STRUC
RECOVER CFSTRUCT

MQCMD_REFRESH_CLUSTER
REFRESH CLUSTER

MQCMD_REFRESH_Q_MGR
REFRESH QMGR

MQCMD_REFRESH_SECURITY
REFRESH SECURITY

MQCMD_RESET_CHANNEL
RESET CHANNEL

MQCMD_RESET_CLUSTER
RESET CLUSTER

MQCMD_RESET_Q_MGR
RESET QMGR

MQCMD_RESET_Q_STATS
RESET QSTATS

MQCMD_RESET_TPIPE
RESET TPIPE

MQCMD_RESOLVE_CHANNEL
RESOLVE CHANNEL

MQCMD_RESOLVE_INDOUBT
RESOLVE INDOUBT

MQCMD_RESUME_Q_MGR
RESUME QMGR other than CLUSTER/CLUSNL

MQCMD_RESUME_Q_MGR_CLUSTER
RESUME QMGR CLUSTER/CLUSNL

MQCMD_REVERIFY_SECURITY
REVERIFY SECURITY

MQCMD_SET_ARCHIVE
SET ARCHIVE

MQCMD_SET_CHLAUTH_REC
SET CHLAUTH

MQCMD_SET_LOG
SET LOG

MQCMD_SET_SYSTEM
SET SYSTEM

MQCMD_START_CHANNEL
START CHANNEL

MQCMD_START_CHANNEL_INIT
START CHINIT

MQCMD_START_CHANNEL_LISTENER
START LISTENER

MQCMD_START_CMD_SERVER
START CMDSERV

MQCMD_START_SERVICE
START SERVICE

MQCMD_START_TRACE
START TRACE

MQCMD_STOP_CHANNEL
STOP CHANNEL

MQCMD_STOP_CHANNEL_INIT
STOP CHINIT

MQCMD_STOP_CHANNEL_LISTENER
STOP LISTENER

MQCMD_STOP_CMD_SERVER
STOP CMDSERV

MQCMD_STOP_CONNECTION
STOP CONN

MQCMD_STOP_SERVICE
STOP SERVICE

MQCMD_STOP_TRACE
STOP TRACE

MQCMD_SUSPEND_Q_MGR
SUSPEND QMGR other than CLUSTER/CLUSNL

MQCMD_SUSPEND_Q_MGR_CLUSTER
SUSPEND QMGR CLUSTER/CLUSNL

Returned: Always.

CommandData

Description: PCF group containing the elements related to the command data.
Identifier: MQGACF_COMMAND_DATA.
Data type: MQCFGR.
PCF elements in group:
- If generated for an MQSC command, this group only contains the PCF element *CommandMQSC*.
- If generated for a PCF command, this group contains the PCF elements that make up the PCF command, exactly as in the command message.

Returned: Always.

CommandMQSC

Description: The text of the MQSC command.
Identifier: MQCACF_COMMAND_MQSC.
Data type: MQCFST.
Maximum length: MQ_COMMAND_MQSC_LENGTH.
Returned: Only if Reason in the message descriptor is MQRC_COMMAND_MQSC.

Create object:

Event name:	Create object.
Reason code in MQCFH:	MQRC_CONFIG_CREATE_OBJECT (2367, X'93F'). New object created.
Event description:	A DEFINE or DEFINE REPLACE command was issued which successfully created a new object.
Event type:	Configuration.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Event data

EventUserId

Description: The user id that issued the command or call that generated the event. (This is the same user id that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MQSET
MQSET call.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Data type: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Data type: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Data type: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Data type: MQCFIN.

Values:

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication record.

MQOT_NAMELIST
Namelist.

MQOT_NONE
No object.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

MQOT_COMM_INFO
Communication information.

MQOT_LISTENER
Channel Listener.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_STORAGE_CLASS
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_NAME
- MQCA_COMM_INFO_NAME
- MQCACH_LISTENER_NAME

Note: MQCACH_CHANNEL_NAME can also be used for channel authentication.

Data type: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.
Returned: Always

Disposition

Description:	Object disposition:
Identifier:	MQIA_QSG_DISP.
Data type:	MQCFIN.
Values:	**MQQSGD_Q_MGR** Object resides on page set of queue manager. **MQQSGD_SHARED** Object resides in shared repository and messages are shared in coupling facility. **MQQSGD_GROUP** Object resides in shared repository. **MQQSGD_COPY** Object resides on page set of queue manager and is a local copy of a GROUP object.
Returned:	Always, except for CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see “Object attributes for event data” on page 2872

Default Transmission Queue Type Error:

Event name:	Default Transmission Queue Type Error.
Reason code in MQCFH:	MQRC_DEF_XMIT_Q_TYPE_ERROR (2198, X'896'). Default transmission queue not local.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank. No transmission queue is defined with the same name as the destination queue manager, so the local queue manager has attempted to use the default transmission queue. However, although there is a queue defined by the *DefXmitQName* queue-manager attribute, it is not a local queue. See Defining a transmission queue for more information about transmission queues.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Default transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

QType

Description: Type of default transmission queue.
Identifier: MQIA_Q_TYPE.
Data type: MQCFIN.
Values:

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Related information:

Defining a transmission queue

DefXmitQName (MQCHAR48)

DefaultTransmissionQueueName property

Default Transmission Queue Usage Error:

Event name:	Default Transmission Queue Usage Error.
Reason code in MQCFH:	MQRC_DEF_XMIT_Q_USAGE_ERROR (2199, X'897'). Default transmission queue usage error.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank. No transmission queue is defined with the same name as the destination queue manager, so the local queue manager has attempted to use the default transmission queue. However, the queue defined by the *DefXmitQName* queue-manager attribute does not have a *Usage* attribute of MQUS_TRANSMISSION. See the Defining a transmission queue for more information about default transmission queues.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Default transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Related information:

Defining a transmission queue

DefXmitQName (MQCHAR48)

DefaultTransmissionQueueName property

Delete object:

Event name:	Delete object.
Reason code in MQCFH:	MQRC_CONFIG_DELETE_OBJECT (2369, X'941'). Object deleted.
Event description:	A DELETE command or MQCLOSE call was issued that successfully deleted an object.
Event type:	Configuration.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Event data

EventUserId

Description:	The user id that issued the command or call that generated the event. (This is the same user id that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier:	MQCACF_EVENT_USER_ID.
Data type:	MQCFST.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Data type: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Data type: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Data type: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Data type: MQCFIN.

Values:

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication record.

MQOT_NAMELIST
Namelist.

MQOT_NONE
No object.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

MQOT_COMM_INFO
Communication information.

MQOT_LISTENER
Channel Listener.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_STORAGE_CLASS
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_NAME
- MQCA_COMM_INFO_NAME
- MQCACH_LISTENER_NAME

Note: MQCACH_CHANNEL_NAME can also be used for channel authentication.

Data type: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.
Returned: Always

Disposition

Description:	Object disposition:
Identifier:	MQIA_QSG_DISP.
Data type:	MQCFIN.
Values:	**MQQSGD_Q_MGR** Object resides on page set of queue manager. **MQQSGD_SHARED** Object resides in shared repository and messages are shared in coupling facility. **MQQSGD_GROUP** Object resides in shared repository. **MQQSGD_COPY** Object resides on page set of queue manager and is a local copy of a GROUP object.
Returned:	Always, except for CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 2872.

Get Inhibited:

Event name:	Get Inhibited.
Reason code in MQCFH:	MQRC_GET_INHIBITED (2016, X'7E0'). Gets inhibited for the queue.
Event description:	MQGET calls are currently inhibited for the queue (see InhibitGet (MQLONG) for the *InhibitGet* queue attribute) or for the queue to which this queue resolves.
Event type:	Inhibit.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application that issued the get.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that issued the get.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Related information:
Setting queue attributes
InhibitGet property

Logger:

Event name:	Logger.
Reason code in MQCFH:	MQRC_LOGGER_STATUS (2411, X'96B') New log extent started.
Event description:	Issued when a queue manager starts writing to a new log extent.
Event type:	Logger.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.LOGGER.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

CurrentLogExtent

Description:	Name of the log extent.
Identifier:	MQCACF_CURRENT_LOG_EXTENT_NAME.
Data type:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

RestartRecoveryLogExtent

Description:	Name of the oldest log extent.
Identifier:	MQCACF_RESTART_LOG_EXTENT_NAME.
Data type:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

MediaRecoveryLogExtent

Description:	Name of the oldest log extent.
Identifier:	MQCACF_MEDIA_LOG_EXTENT_NAME.
Data type:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

LogPath

Description:	The directory where log files are created by the queue manager.
Identifier:	MQCACF_LOG_PATH.
Data type:	MQCFST.
Maximum length:	MQ_LOG_PATH_LENGTH.
Returned:	Always.

Not Authorized (type 1):

Event name:	Not Authorized (type 1).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQCONN or system connection call, the user is not authorized to connect to the queue manager. *ReasonQualifier* identifies the nature of the error.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 1 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_CONN_NOT_AUTHORIZED
Connection not authorized.

MQRQ_SYS_CONN_NOT_AUTHORIZED
Missing system authority.

Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application causing the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application causing the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Not Authorized (type 2):

Event name:	Not Authorized (type 2).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQOPEN or MQPUT1 call, the user is not authorized to open the object for the options specified.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier for type 2 authority events.	
Identifier:	MQIACF_REASON_QUALIFIER.	
Data type:	MQCFIN.	
Values:	MQRQ_OPEN_NOT_AUTHORIZED	Open not authorized.
Returned:	Always.	

Options

Description: Options specified on the MQOPEN call.
Identifier: MQIACF_OPEN_OPTIONS.
Data type: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application that caused the authorization check.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that caused the authorization check.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Object queue manager name from object descriptor (MQOD).
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectQMgrName* in the object descriptor (MQOD) when the object was opened is not the queue manager currently connected.

QName

Description: Object name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: If the object opened is a queue object.

ProcessName

Description: Name of process object from object descriptor (MQOD).
Identifier: MQCA_PROCESS_NAME.
Data type: MQCFST.
Maximum length: MQ_PROCESS_NAME_LENGTH.
Returned: If the object opened is a process object.

TopicString

Description: Topic string being subscribed to, or opened.
Identifier: MQCA_TOPIC_STRING.
Data type: MQCFST.
Maximum length: MQ_TOPIC_STR_LENGTH.
Returned: If the object opened is a topic object.

AdminTopicNames

Description: List of topic admin objects against which authority is checked.
Identifier: MQCA_ADMIN_TOPIC_NAME.
Data type: MQCFSL.
Maximum length: MQ_TOPIC_NAME_LENGTH.
Returned: If the object opened is a topic object.

ObjectType

Description: Object type from object descriptor (MQOD).
Identifier: MQIACF_OBJECT_TYPE.
Data type: MQCFIN.
Values:
- MQOT_NAMELIST Namelist.
- MQOT_PROCESS Process.
- MQOT_Q Queue.
- MQOT_Q_MGR Queue manager.
- MQOT_TOPIC Topic.

Returned: Always.

NamelistName

Description: Object name from object descriptor (MQOD).
Identifier: MQCA_NAMELIST_NAME.
Data type: MQCFST.
Maximum length: MQ_NAMELIST_NAME_LENGTH.
Returned: If the object opened is a namelist object.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Not Authorized (type 3):

Event name:	Not Authorized (type 3).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	When closing a queue using the MQCLOSE call, the user is not authorized to delete the object, which is a permanent dynamic queue, and the *Hobj* parameter specified on the MQCLOSE call is not the handle returned by the MQOPEN call that created the queue. When closing a subscription using an MQCLOSE call, the user has requested that the subscription is removed using the MQCO_REMOVE_SUB option, but the user is not the creator of the subscription or does not have *sub* authority on the topic associated with the subscription.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description: Identifier for type 3 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_CLOSE_NOT_AUTHORIZED
Close not authorized.

Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check
Identifier: MQCACF_USER_IDENTIFIER
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application causing the authorization check.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application causing the authorization check.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

QName

Description: Object name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: If the handle being closed is a queue

SubName

Description: Name of subscription being removed.
Identifier: MQCACF_SUB_NAME.
Data type: MQCFST.
Maximum length: MQ_SUB_NAME_LENGTH.
Returned: If the handle being closed is a subscription.

TopicString

Description: Topic string of the subscription.
Identifier: MQCA_TOPIC_STRING
Data type: MQCFST.
Maximum length: MQ_TOPIC_STR_LENGTH.
Returned: If the handle being closed is a subscription.

AdminTopicNames

Description: List of topic administration objects against which authority was checked.
Identifier: MQCA_ADMIN_TOPIC_NAME
Data type: MQCFSL.
Maximum length: MQ_TOPIC_NAME_LENGTH.
Returned: If the handle being closed is a subscription.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Not Authorized (type 4):

Event name:	Not Authorized (type 4).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	Indicates that a command has been issued from a user ID that is not authorized to access the object specified in the command.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 4 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_CMD_NOT_AUTHORIZED
Command not authorized.

Returned: Always.

Command

Description: Command identifier. See the MQCFH header structure, described in "Event message MQCFH (PCF header)" on page 2913.
Identifier: MQIACF_COMMAND.
Data type: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

Not Authorized (type 5):

Event name:	Not Authorized (type 5).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQSUB call, the user is not authorized to subscribe to the specified topic.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 5 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_SUB_NOT_AUTHORIZED
Subscribe not authorized.

Returned: Always.

Options

Description: Options specified on the MQSUB call.
Identifier: MQIACF_SUB_OPTIONS
Data type: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application that caused the authorization check.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that caused the authorization check.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

TopicString

Description:	Topic string being opened or subscribed to.
Identifier:	MQCA_TOPIC_STRING.
Data type:	MQCFST.
Maximum length:	MQ_TOPIC_STR_LENGTH.
Returned:	Always.

AdminTopicNames

Description:	List of topic administration objects against which authority is checked.
Identifier:	MQCA_ADMIN_TOPIC_NAME.
Data type:	MQCFSL.
Maximum length:	MQ_TOPIC_NAME_LENGTH.
Returned:	Always.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Not Authorized (type 6):

Event name:	Not Authorized (type 6).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQSUB call, the user is not authorized to use the destination queue with the required level of access. This event is only returned for subscriptions using non-managed destination queues. When creating, altering, or resuming a subscription, and a handle to the destination queue is supplied on the request, the user does not have PUT authority on the destination queue provided. When resuming or alerting a subscription and the handle to the destination queue is to be returned on the MQSUB call, and the user does not have PUT, GET and BROWSE authority on the destination queue.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 6 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Data type: MQCFIN.
Values:

MQRQ_SUB_DEST_NOT_AUTHORIZED
Subscription destination queue usage not authorized.

Returned: Always.

Options

Description: Options specified on the MQSUB call.
Identifier: MQIACF_SUB_OPTIONS
Data type: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application that caused the authorization check.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that caused the authorization check.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

TopicString

Description: Topic string being subscribed to.
Identifier: MQCA_TOPIC_STRING.
Data type: MQCFST.
Maximum length: MQ_TOPIC_STR_LENGTH.
Returned: Always.

DestQMgrName

Description: Hosting queue manager name of the subscription's destination queue.
Identifier: MQCACF_OBJECT_Q_MGR_NAME
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the queue manager hosting the destination queue is not the queue manager to which the application is currently connected.

DestQName

Description: The name of the destination queue of the subscription..
Identifier: MQCA_Q_NAME
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

DestOpenOptions

Description: The open options requested for the destination queue.
Identifier: MQIACF_OPEN_OPTIONS
Data type: MQCFIN.
Returned: Always.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Put Inhibited:

Event name:	Put Inhibited.
Reason code in MQCFH:	MQRC_PUT_INHIBITED (2051, X'803'). Put calls inhibited for the queue or topic.
Event description:	MQPUT and MQPUT1 calls are currently inhibited for the queue or topic (see the *InhibitPut* queue attribute in InhibitPut (MQLONG) or the *InhibitPublications* topic attribute in "Topic attributes" on page 2902 for the queue to which this queue resolves.
Event type:	Inhibit.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: If the object opened is a queue object

ApplType

Description: Type of application that issued the put.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application that issued the put.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of queue manager from object descriptor (MQOD).
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Only if this parameter has a value different from *QMgrName*. This occurs when the *ObjectQMgrName* field in the object descriptor provided by the application on the MQOPEN or MQPUT1 call is neither blank nor the name of the application's local queue manager. However, it can also occur when *ObjectQMgrName* in the object descriptor is blank, but a name service provides a queue-manager name that is not the name of the application's local queue manager.

TopicString

Description: Topic String being opened
Identifier: MQCA_TOPIC_STRING
Data type: MQCFST.
Maximum length: MQ_TOPIC_STR_LENGTH.
Returned: If the object opened is a topic.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Related information:

InhibitPut property

Inquire Queue (Response)
The response to the Inquire Queue command MQCMD_INQUIRE_Q consists of the response header followed by the *QName* structure. On z/OS only, response includes the *QSGDisposition* structure, and the requested combination of attribute parameter structures.

Inquire Topic (Response)
The response to the Inquire Topic (MQCMD_INQUIRE_TOPIC) command consists of the response header followed by the *TopicName* structure (and on z/OS only, the *QSG Disposition* structure), and the requested combination of attribute parameter structures (where applicable).

Inquire Topic Status (Response)
The response of the Inquire topic (MQCMD_INQUIRE_TOPIC_STATUS) command consists of the response header, followed by the *TopicString* structure, and the requested combination of attribute parameter structures (where applicable). The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS. The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS_SUB. The Inquire Topic Status command returns the values requested when the *StatusType* is MQIACF_TOPIC_STATUS_PUB.

Change, Copy, and Create Topic
The Change Topic command changes existing topic definitions. The Copy and Create Topic commands create new topic definitions - the Copy command uses attribute values of an existing topic definition.

Queue Depth High:

Event name:	Queue Depth High.
Reason code in MQCFH:	MQRC_Q_DEPTH_HIGH (2224, X'8B0'). Queue depth high limit reached or exceeded.
Event description:	An MQPUT or MQPUT1 call has caused the queue depth to be incremented to or above the limit specified in the *QDepthHighLimit* attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

1. WebSphere MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.
2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "Event message MQMD (message descriptor)" on page 2908 for more information.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Name of the queue on which the limit has been reached.
Identifier:	MQCA_BASE_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset. The value recorded by this timer is also used as the *interval time* in queue service interval events.
Identifier:	MQIA_TIME_SINCE_RESET.
Data type:	MQCFIN.
Returned:	Always.

HighQDepth

Description:	Maximum number of messages on the queue since the queue statistics were last reset.
Identifier:	MQIA_HIGH_Q_DEPTH.
Data type:	MQCFIN.
Returned:	Always.

MsgEnqCount

Description:	Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier:	MQIA_MSG_ENQ_COUNT.
Data type:	MQCFIN.
Returned:	Always.

MsgDeqCount

Description:	Number of messages removed from the queue since the queue statistics were last reset.
Identifier:	MQIA_MSG_DEQ_COUNT.
Data type:	MQCFIN.
Returned:	Always.

Queue Depth Low:

Event name:	Queue Depth Low.
Reason code in MQCFH:	MQRC_Q_DEPTH_LOW (2225, X'8B1'). Queue depth low limit reached or exceeded.
Event description:	A get operation has caused the queue depth to be decremented to or below the limit specified in the *QDepthLowLimit* attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

1. WebSphere MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.
2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "Event message MQMD (message descriptor)" on page 2908 for more information.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Name of the queue on which the limit has been reached.
Identifier: MQCA_BASE_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

TimeSinceReset

Description: Time, in seconds, since the statistics were last reset. The value recorded by this timer is also used as the *interval time* in queue service interval events.
Identifier: MQIA_TIME_SINCE_RESET.
Data type: MQCFIN.
Returned: Always.

HighQDepth

Description: Maximum number of messages on the queue since the queue statistics were last reset.
Identifier: MQIA_HIGH_Q_DEPTH.
Data type: MQCFIN.
Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Data type: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Data type: MQCFIN.
Returned: Always.

Queue Full:

Event name:	Queue Full.
Reason code in MQCFH:	MQRC_Q_FULL (2053, X'805'). Queue already contains maximum number of messages.
Event description:	On an MQPUT or MQPUT1 call, the call failed because the queue is full. That is, it already contains the maximum number of messages possible (see the *MaxQDepth* local-queue attribute
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

1. WebSphere MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.
2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "Event message MQMD (message descriptor)" on page 2908 for more information.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Name of the queue on which the put was rejected.
Identifier:	MQCA_BASE_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset.
Identifier:	MQIA_TIME_SINCE_RESET.
Data type:	MQCFIN.
Returned:	Always.

HighQDepth

Description: Maximum number of messages on a queue.
Identifier: MQIA_HIGH_Q_DEPTH.
Data type: MQCFIN.
Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Data type: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Data type: MQCFIN.
Returned: Always.

Queue Manager Active:

Event name:	Queue Manager Active.
Reason code in MQCFH:	MQRC_Q_MGR_ACTIVE (2222, X'8AE'). Queue manager active.
Event description:	This condition is detected when a queue manager becomes active.
Event type:	Start And Stop.
Platforms:	All, except the first start of a WebSphere MQ for z/OS queue manager. In this case it is produced only on subsequent restarts.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

Queue Manager Not Active:

Event name:	Queue Manager Not Active.
Reason code in MQCFH:	MQRC_Q_MGR_NOT_ACTIVE (2223, X'8AF'). Queue manager unavailable.
Event description:	This condition is detected when a queue manager is requested to stop or quiesce.
Event type:	Start And Stop.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier of causes of this reason code. This specifies the type of stop that was requested.
Identifier:	MQIACF_REASON_QUALIFIER.
Data type:	MQCFIN.
Values:	**MQRQ_Q_MGR_STOPPING** Queue manager stopping. **MQRQ_Q_MGR_QUIESCING** Queue manager quiescing.
Returned:	Always.

Queue Service Interval High:

Event name:	Queue Service Interval High.
Reason code in MQCFH:	MQRC_Q_SERVICE_INTERVAL_HIGH (2226, X'8B2'). Queue service interval high.
Event description:	No successful get operations or MQPUT calls have been detected within an interval greater than the limit specified in the *QServiceInterval* attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note: WebSphere MQ for z/OS does not support service interval events on shared queues.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Name of the queue specified on the command that caused this queue service interval event to be generated.
Identifier: MQCA_BASE_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

TimeSinceReset

Description: Time, in seconds, since the statistics were last reset. For a service interval high event, this value is greater than the service interval.
Identifier: MQIA_TIME_SINCE_RESET.
Data type: MQCFIN.
Returned: Always.

HighQDepth

Description: Maximum number of messages on the queue since the queue statistics were last reset.
Identifier: MQIA_HIGH_Q_DEPTH.
Data type: MQCFIN.
Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Data type: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Data type: MQCFIN.
Returned: Always.

Queue Service Interval OK:

Event name:	Queue Service Interval OK.
Reason code in MQCFH:	MQRC_Q_SERVICE_INTERVAL_OK (2227, X'8B3'). Queue service interval OK.
Event description:	A successful get operation has been detected within an interval less than or equal to the limit specified in the *QServiceInterval* attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note: WebSphere MQ for z/OS does not support service interval events on shared queues.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name specified on the command that caused this queue service interval event to be generated.
Identifier:	MQCA_BASE_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset.
Identifier:	MQIA_TIME_SINCE_RESET.
Data type:	MQCFIN.
Returned:	Always.

HighQDepth

Description:	Maximum number of messages on the queue since the queue statistics were last reset.
Identifier:	MQIA_HIGH_Q_DEPTH.
Data type:	MQCFIN.
Returned:	Always.

MsgEnqCount

Description:	Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier:	MQIA_MSG_ENQ_COUNT.
Data type:	MQCFIN.
Returned:	Always.

MsgDeqCount

Description:	Number of messages removed from the queue since the queue statistics were last reset.
Identifier:	MQIA_MSG_DEQ_COUNT.
Data type:	MQCFIN.
Returned:	Always.

Queue Type Error:

Event name:	Queue Type Error.
Reason code in MQCFH:	MQRC_Q_TYPE_ERROR (2057, X'809'). Queue type not valid.
Event description:	On an MQOPEN call, the *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue (in order to specify a queue-manager alias). In that local definition the *RemoteQMgrName* attribute is the name of the local queue manager. However, the *ObjectName* field specifies the name of a model queue on the local queue manager, which is not allowed. See the Queue manager events for more information.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Always.

ApplName

Description:	Name of the application making the MQI call that caused the event.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Refresh object:

Event name:	Refresh object.
Reason code in MQCFH:	MQRC_CONFIG_REFRESH_OBJECT (2370, X'942'). Refresh queue manager configuration.
Event description:	A REFRESH QMGR command specifying TYPE (CONFIGEV) was issued.
Event type:	Configuration.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Note: The REFRESH QMGR command can produce many configuration events; one event is generated for each object that is selected by the command.

Event data

EventUserId

Description: The user id that issued the command or call that generated the event. (This is the same user id that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MQMD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Data type: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Data type: MQCFIN.
Values:

MQEVO_CONSOLE
Console command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

MQEVO_OTHER
None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MQMD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MQMD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Data type: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Data type: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MQMD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Data type: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MQMD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Data type: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Data type: MQCFIN.

Values:

MQOT_CHANNEL
Channel.

MQOT_CHLAUTH
Channel authentication record.

MQOT_NAMELIST
Namelist.

MQOT_NONE
No object.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_Q_MGR
Queue manager.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

MQOT_COMM_INFO
Communication information.

MQOT_LISTENER
Channel Listener.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_Q_MGR_NAME
- MQCA_STORAGE_CLASS
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_NAME
- MQCA_COMM_INFO_NAME
- MQCACH_LISTENER_NAME

Note: MQCACH_CHANNEL_NAME can also be used for channel authentication.

Data type: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.
Returned: Always

Disposition

Description:	Object disposition:
Identifier:	MQIA_QSG_DISP.
Data type:	MQCFIN.
Values:	**MQQSGD_Q_MGR** Object resides on page set of queue manager. **MQQSGD_SHARED** Object resides in shared repository and messages are shared in coupling facility. **MQQSGD_GROUP** Object resides in shared repository. **MQQSGD_COPY** Object resides on page set of queue manager and is a local copy of a GROUP object.
Returned:	Always, except for queue manager and CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see "Object attributes for event data" on page 2872.

Remote Queue Name Error:

Event name:	Remote Queue Name Error.
Reason code in MQCFH:	MQRC_REMOTE_Q_NAME_ERROR (2184, X'888'). Remote queue name not valid.
Event description:	On an MQOPEN or MQPUT1 call one of the following occurs: • A local definition of a remote queue (or an alias to one) was specified, but the *RemoteQName* attribute in the remote queue definition is blank. Note that this error occurs even if the *XmitQName* in the definition is not blank. • The *ObjectQMgrName* field in the object descriptor is not blank and not the name of the local queue manager, but the *ObjectName* field is blank.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Transmission Queue Type Error:

Event name:	Transmission Queue Type Error.
Reason code in MQCFH:	MQRC_XMIT_Q_TYPE_ERROR (2091, X'82B'). Transmission queue not local.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition. Either: • *XmitQName* is not blank, but specifies a queue that is not a local queue, or • *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that is not a local queue This also occurs if the queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a queue, but this is not a local queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCA_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

QType

Description: Type of transmission queue.
Identifier: MQIA_Q_TYPE.
Data type: MQCFIN.
Values:

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Transmission Queue Usage Error:

Event name:	Transmission Queue Usage Error.
Reason code in MQCFH:	MQRC_XMIT_Q_USAGE_ERROR (2092, X'82C'). Transmission queue with wrong usage.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager, but one of the following occurred. Either: • *ObjectQMgrName* specifies the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION. • The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition: – *XmitQName* is not blank, but specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION – *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION • The queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Unknown Alias Base Queue:

Event name:	Unknown Alias Base Queue.
Reason code in MQCFH:	MQRC_UNKNOWN_ALIAS_BASE_Q (2082, X'822'). Unknown alias base queue or topic.
Event description:	An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseObjectName* in the alias queue attributes is not recognized as a queue or topic name.
Event type:	Local.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

BaseObjectName

Description:	Object name to which the alias resolves.
Identifier:	MQCA_BASE_OBJECT_NAME. For compatibility with existing applications, you can still use MQCA_BASE_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

BaseType

Description: Type of object to which the alias resolves.
Identifier: MQIA_BASE_TYPE.
Data type: MQCFIN.
Values:

MQOT_Q
Base object type is a queue

MQOT_TOPIC
Base object type is a topic

Returned: Always.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Unknown Default Transmission Queue:

Event name:	Unknown Default Transmission Queue.
Reason code in MQCFH:	MQRC_UNKNOWN_DEF_XMIT_Q (2197, X'895'). Unknown default transmission queue.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. If a local definition of the remote queue was specified, or if a queue-manager alias is being resolved, the *XmitQName* attribute in the local definition is blank. No queue is defined with the same name as the destination queue manager. The queue manager has therefore attempted to use the default transmission queue. However, the name defined by the *DefXmitQName* queue-manager attribute is not the name of a locally-defined queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Default transmission queue name.
Identifier:	MQCA_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description: Type of application attempting to open the remote queue.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application attempting to open the remote queue.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Unknown Object Name:

Event name:	Unknown Object Name.
Reason code in MQCFH:	MQRC_UNKNOWN_OBJECT_NAME (2085, X'825'). Unknown object name.
Event description:	On an MQOPEN or MQPUT1 call, the *ObjectQMgrName* field in the object descriptor MQOD is set to one of the following options. It is either: • Blank • The name of the local queue manager • The name of a local definition of a remote queue (a queue-manager alias) in which the *RemoteQMgrName* attribute is the name of the local queue manager However, the *ObjectName* in the object descriptor is not recognized for the specified object type.
Event type:	Local.

Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Always.

ApplName

Description:	Name of the application making the MQI call that caused the event.
Identifier:	MQCACF_APPL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	If the object opened is a queue object. Either *QName* or *TopicName* is returned.

ProcessName

Description:	Process object name from object descriptor (MQOD).
Identifier:	MQCA_PROCESS_NAME.
Data type:	MQCFST.
Maximum length:	MQ_PROCESS_NAME_LENGTH.
Returned:	If the object opened is a process object. One of *ProcessName*, *QName*, or *TopicName* is returned.

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

TopicName

Description:	Topic object name from object descriptor (MQOD).
Identifier:	MQCA_TOPIC_NAME.
Data type:	MQCFST.
Maximum length:	MQ_TOPIC_NAME_LENGTH.
Returned:	If the object opened is a topic object. One of *ProcessName*, *QName*, or *TopicName* is returned.

ConnName

Description:	Connection name for client connection.
Identifier:	MQCACH_CONNECTION_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description:	Channel name for client connection.
Identifier:	MQCACH_CHANNEL_NAME.
Data type:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	If the application making the MQI call that caused the event is a client attached application.

Unknown Remote Queue Manager:

Event name:	Unknown Remote Queue Manager.
Reason code in MQCFH:	MQRC_UNKNOWN_REMOTE_Q_MGR (2087, X'827'). Unknown remote queue manager.

Event description:	On an MQOPEN or MQPUT1 call, an error occurred with queue-name resolution, for one of the following reasons: • *ObjectQMgrName* is either blank or the name of the local queue manager, and *ObjectName* is the name of a local definition of a remote queue that has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank. • *ObjectQMgrName* is the name of a queue-manager alias definition (held as the local definition of a remote queue) that has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank. • *ObjectQMgrName* specified is not: – Blank – The name of the local queue manager – The name of a local queue – The name of a queue-manager alias definition (that is, a local definition of a remote queue with a blank *RemoteQName*) and the *DefXmitQName* queue-manager attribute is blank. • *ObjectQMgrName* is blank or is the name of the local queue manager, and *ObjectName* is the name of a local definition of a remote queue (or an alias to one), for which *RemoteQMgrName* is either blank or is the name of the local queue manager. This error occurs even if the *XmitQName* is not blank. • *ObjectQMgrName* is the name of a local definition of a remote queue. In this case, it should be a queue-manager alias definition, but the *RemoteQName* in the definition is not blank. • *ObjectQMgrName* is the name of a model queue. • The queue name is resolved through a cell directory. However, there is no queue defined with the same name as the remote queue manager name obtained from the cell directory. Also, the *DefXmitQName* queue-manager attribute is blank. • On z/OS only: a message was put to a queue manager in a queue-sharing group and *SQQMNAME* is set to USE. This routes the message to the specified queue manager in order to be put on the queue. If *SQQMNAME* is set to IGNORE, the message is put to the queue directly.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application attempting to open the remote queue.
Identifier: MQIA_APPL_TYPE.
Data type: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application attempting to open the remote queue.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Unknown Transmission Queue:

Event name:	Unknown Transmission Queue.
Reason code in MQCFH:	MQRC_UNKNOWN_XMIT_Q (2196, X'894'). Unknown transmission queue.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or the *ObjectQMgrName* in the object descriptor specifies the name of a local definition of a remote queue (in the latter case queue-manager aliasing is being used). However, the *XmitQName* attribute of the definition is not blank and not the name of a locally-defined queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCA_XMIT_Q_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.
Data type:	MQCFIN.
Returned:	Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Data type: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Data type: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

ConnName

Description: Connection name for client connection.
Identifier: MQCACH_CONNECTION_NAME.
Data type: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: If the application making the MQI call that caused the event is a client attached application.

ChannelName

Description: Channel name for client connection.
Identifier: MQCACH_CHANNEL_NAME.
Data type: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH
Returned: If the application making the MQI call that caused the event is a client attached application.

Troubleshooting and support reference

Use the reference information in this section to help you diagnose errors with IBM WebSphere MQ.

Select the appropriate topic from the following list to diagnose problems and errors in IBM WebSphere MQ:

- "An example of WebSphere MQ for Windows trace data"
- "Example trace data for WebSphere MQ for UNIX and Linux systems" on page 3009
- "Examples of trace output" on page 3013
- "Examples of CEDF output" on page 3015

Related information:

Troubleshooting and support

Troubleshooting overview

Using trace

An example of WebSphere MQ for Windows trace data

An extract from a WebSphere MQ for Windows trace file.

```
Counter  TimeStamp          PID.TID    Ident        Data
============================================================

00000EF7 16:18:56.381367   2512.1           :       !! - Thread stack
00000EF8 16:18:56.381406   2512.1           :       !! - -> InitProcessInitialisation
00000EF9 16:18:56.381429   2512.1           :       --{  InitProcessInitialisation
00000EFA 16:18:56.381514   2512.1           :       ---{  xcsReleaseThreadMutexSem
00000EFB 16:18:56.381529   2512.1           :       ---}  xcsReleaseThreadMutexSem (rc=OK)
00000EFC 16:18:56.381540   2512.1           :       ---{  xcsGetEnvironmentString
00000EFD 16:18:56.381574   2512.1           :       xcsGetEnvironmentString[AMQ_REUSE_SHARED_THREAD] = NULL
00000EFE 16:18:56.381587   2512.1           :       ---}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000EFF 16:18:56.381612   2512.1           :       ---{  xcsGetEnvironmentInteger
00000F00 16:18:56.381622   2512.1           :       ----{  xcsGetEnvironmentString
00000F01 16:18:56.381647   2512.1           :       xcsGetEnvironmentString[AMQ_AFFINITY_MASK] = NULL
00000F02 16:18:56.381660   2512.1           :       ----}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F03 16:18:56.381673   2512.1           :       ---}! xcsGetEnvironmentInteger (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F04 16:18:56.381684   2512.1           :       ---{  xcsGetEnvironmentString
00000F05 16:18:56.381708   2512.1           :       xcsGetEnvironmentString[AMQ_FFSTINFO] = NULL
00000F06 16:18:56.381747   2512.1           :       ---}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F07 16:18:56.381760   2512.1           :       ---{  xcsIsEnvironment
00000F08 16:18:56.381783   2512.1           :       xcsIsEnvironment[AMQ_DEBUG_MTIME] = FALSE
00000F09 16:18:56.381793   2512.1           :       ---}  xcsIsEnvironment (rc=OK)
00000F0A 16:18:56.381804   2512.1           :       ---{  xcsGetEnvironmentInteger
00000F0B 16:18:56.381811   2512.1           :       ----{  xcsGetEnvironmentString
00000F0C 16:18:56.381835   2512.1           :       xcsGetEnvironmentString[AMQ_CBM_REUSE_FACTOR] = NULL
00000F0D 16:18:56.381848   2512.1           :       ----}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F0E 16:18:56.381861   2512.1           :       ---}! xcsGetEnvironmentInteger (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F0F 16:18:56.381874   2512.1           :       ---{  xcsGetEnvironmentInteger
00000F10 16:18:56.381885   2512.1           :       ----{  xcsGetEnvironmentString
00000F11 16:18:56.381908   2512.1           :       xcsGetEnvironmentString[AMQ_CBM_MAX_CACHEABLE_SIZE] = NULL
00000F12 16:18:56.381919   2512.1           :       ----}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F13 16:18:56.381929   2512.1           :       ---}! xcsGetEnvironmentInteger (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F14 16:18:56.381941   2512.1           :       ---{  xcsGetEnvironmentInteger
00000F15 16:18:56.381952   2512.1           :       ----{  xcsGetEnvironmentString
00000F16 16:18:56.381976   2512.1           :       xcsGetEnvironmentString[AMQ_CBM_LEN] = NULL
00000F17 16:18:56.381992   2512.1           :       ----}! xcsGetEnvironmentString (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F18 16:18:56.382003   2512.1           :       ---}! xcsGetEnvironmentInteger (rc=xecE_E_ENV_VAR_NOT_FOUND)
00000F19 16:18:56.382016   2512.1           :       --}  InitProcessInitialisation (rc=OK)
00000F1A 16:18:56.383045   2512.1           :       --{  DLLMain
00000F1B 16:18:56.383059   2512.1           :       ---{  MCSInitCriticalSection
00000F1C 16:18:56.383068   2512.1           :       ---}  MCSInitCriticalSection (rc=OK)
```

Figure 79. Sample WebSphere MQ for Windows trace

Example trace data for WebSphere MQ for UNIX and Linux systems

An extract from a WebSphere MQ for HP-UX trace file.

```
Timestamp        Process.Thread  Trace Ident  Trace Data
=========================================================
 10:36:38.973286    11352.1           :           Header.v02:7.0:HP-UX B.11.31:64:0:1:GMT
 10:36:38.973328    11352.1           :           Version : 7.0.1.3    Level : p701-103-100814
 10:36:38.973347    11352.1           :           UTC   Date : 02/28/12  Time : 10:36:38.973271
 10:36:38.973356    11352.1           :           Local Date : 02/28/12  Time : 10:36:38.973271  GMT
 10:36:38.973378    11352.1           :           PID : 11352 Process : dltmqm_nd (64-bit)
 10:36:38.973384    11352.1           :           Host : myhost
 10:36:38.973389    11352.1           :           Operating System : HP-UX B.11.31
 10:36:38.973394    11352.1           :           Product Long Name : WebSphere MQ for HP-UX (Itanium platform)
 10:36:38.973399    11352.1           :           --------------------------------
 10:36:38.973405    11352.1           :           xtrNullFd: 4, xihTraceFileNum: 5
 10:36:38.973434    11352.1           :           Thread stack
 10:36:38.974303    11352.1           :           -> InitProcessInitialisation
 10:36:38.974309    11352.1           :      {  InitProcessInitialisation
 10:36:38.974314    11352.1           :      -{  xcsIsEnvironment
 10:36:38.974338    11352.1           :           xcsIsEnvironment[AMQ_NO_CS_RELOAD] = FALSE
 10:36:38.974343    11352.1           :      -}  xcsIsEnvironment rc=OK
 10:36:38.974356    11352.1           :      -{  xcsLoadFunction
 10:36:38.974362    11352.1           :           LibName(libmqmcs_r.so) LoadType(2097200)
 10:36:38.974368    11352.1           :           General, comms, CS, OAM, or WAS
 10:36:38.974388    11352.1           :      --{  xcsQueryValueForSubpool
 10:36:38.974401    11352.1           :      --}  xcsQueryValueForSubpool rc=OK
 10:36:38.974451    11352.1           :           FullPathLibName(/opt/mqm/lib64/libmqmcs_r.so) loaded with dlopen
 10:36:38.974456    11352.1           :      --{  xcsGetMemFn
 10:36:38.974463    11352.1           :           component:24 function:176 length:2088 options:0 cbmindex:-1 *pointer:600
 10:36:38.974468    11352.1           :      --}  xcsGetMemFn rc=OK
 10:36:38.974475    11352.1           :           Handle(0000000000000000) Function(0000000000000000) FullPathLibName(/opt
 10:36:38.974480    11352.1           :      -}  xcsLoadFunction rc=OK
 10:36:38.974486    11352.1           :           SystemPageSize is 4096.
 10:36:38.974493    11352.1           :           getrlimit for RLIMIT_NOFILE returned rlim_cur=2048 rlim_max=4096
```

Figure 80. Sample WebSphere MQ for HP-UX trace

Figure 81 on page 3011 shows an extract from a WebSphere MQ for Solaris trace:

```
Timestamp        Process.Thread  Trace Ident  Trace Data
=========================================================
 11:48:57.905466     7078.1          :           Header.v02:7.0:SunOS 5.9:64:-1:1:GMT
 11:48:57.905625     7078.1          :           Version : 7.0.0.0    Level : p000-L090514
 11:48:57.905770     7078.1          :           UTC   Date : 05/15/09  Time : 10:48:57.905364
 11:48:57.905816     7078.1          :           Local Date : 05/15/09  Time : 11:48:57.905364  GMT
 11:48:57.906104     7078.1          :           PID : 7078 Process : dltmqm_nd (64-bit)
 11:48:57.906129     7078.1          :           Host : computer.v6.hursley.ibm.com
 11:48:57.906148     7078.1          :           Operating System : SunOS 5.9
 11:48:57.906167     7078.1          :           Product Long Name : WebSphere MQ for Solaris (SPARC platform)
 11:48:57.906184     7078.1          :           --------------------------------
 11:48:57.906203     7078.1          :           xtrNullFd: 4, xihTraceFileNum: 5
 11:48:57.906276     7078.1          :           Thread stack
 11:48:57.906353     7078.1          :      {  xcsInitialize
 11:48:57.906385     7078.1          :      -{  InitPrivateServices
 11:48:57.906439     7078.1          :      --{  xcsGetEnvironmentString
 11:48:57.906566     7078.1          :           xcsGetEnvironmentString[MQS_ACTION_ON_EXCEPTION] = NULL
 11:48:57.906608     7078.1          :      --}! xcsGetEnvironmentString rc=xecE_E_ENV_VAR_NOT_FOUND
 11:48:57.906709     7078.1          :      --{  xcsIsEnvironment
 11:48:57.906738     7078.1          :           xcsIsEnvironment[AMQ_SIGCHLD_SIGACTION] = FALSE
 11:48:57.906755     7078.1          :      --}  xcsIsEnvironment rc=OK
 11:48:57.906771     7078.1          :           AMQ_SIGCHLD_SIGACTION is not set
 11:48:57.906835     7078.1          :      --{  xcsIsEnvironment
 11:48:57.906862     7078.1          :           xcsIsEnvironment[MQS_NO_SYNC_SIGNAL_HANDLING] = FALSE
 11:48:57.906878     7078.1          :      --}  xcsIsEnvironment rc=OK
 11:48:57.907000     7078.1          :           FPE Handler installed, New=7e0b0f38, Old=0
 11:48:57.907035     7078.1          :           SEGV Handler installed, New=7e0b0f38, Old=0
 11:48:57.907063     7078.1          :           BUS Handler installed, New=7e0b0f38, Old=0
 11:48:57.907091     7078.1          :           ILL Handler installed, New=7e0b0f38, Old=0
 11:48:57.907109     7078.1          :           Synchronous Signal Handling Activated
```

Figure 81. Sample WebSphere MQ for Solaris trace

Figure 82 on page 3012 shows an extract from a WebSphere MQ for Linux trace:

```
Timestamp        Process.Thread  Trace Ident  Trace Data
=========================================================
 11:02:23.643879     1239.1          :           Header.v02:7.0:Linux 2.6.5-7.276-smp:32:-1:1:GMT
 11:02:23.643970     1239.1          :           Version : 7.0.0.0    Level : p000-L090514
 11:02:23.644025     1239.1          :           UTC   Date : 05/15/09  Time : 10:02:23.643841
 11:02:23.644054     1239.1          :           Local Date : 05/15/09  Time : 11:02:23.643841  GMT
 11:02:23.644308     1239.1          :           PID : 1239 Process : dltmqm (32-bit)
 11:02:23.644324     1239.1          :           Host : hal1
 11:02:23.644334     1239.1          :           Operating System : Linux 2.6.5-7.276-smp
 11:02:23.644344     1239.1          :           Product Long Name : WebSphere MQ for Linux (x86 platform)
 11:02:23.644353     1239.1          :           --------------------------------
 11:02:23.644363     1239.1          :           xtrNullFd: 3, xihTraceFileNum: 4
 11:02:23.644394     1239.1          :           Thread stack
 11:02:23.644412     1239.1          :           -> InitProcessInitialisation
 11:02:23.644427     1239.1          :      {  InitProcessInitialisation
 11:02:23.644439     1239.1          :      -{  xcsIsEnvironment
 11:02:23.644469     1239.1          :           xcsIsEnvironment[AMQ_NO_CS_RELOAD] = FALSE
 11:02:23.644485     1239.1          :      -}  xcsIsEnvironment rc=OK
 11:02:23.644504     1239.1          :      -{  xcsLoadFunction
 11:02:23.644519     1239.1          :           LibName(libmqmcs_r.so) LoadType(2097200)
 11:02:23.644537     1239.1          :           General, comms, CS, OAM, or WAS
 11:02:23.644558     1239.1          :      --{  xcsQueryValueForSubpool
 11:02:23.644579     1239.1          :      --}  xcsQueryValueForSubpool rc=OK
 11:02:23.644641     1239.1          :           FullPathLibName(/opt/mqm/lib/libmqmcs_r.so) loaded with dlopen
 11:02:23.644652     1239.1          :      --{  xcsGetMem
 11:02:23.644675     1239.1          :           component:24 function:176 length:8212 options:0 cbmindex:-1 *pointer:0x8
 11:02:23.644685     1239.1          :      --}  xcsGetMem rc=OK
 11:02:23.644722     1239.1          :           Handle((nil)) Function((nil)) FullPathLibName(/opt/mqm/lib/libmqmcs_r.so
 11:02:23.644732     1239.1          :      -}  xcsLoadFunction rc=OK
 11:02:23.644753     1239.1          :           SystemPageSize is 4096.
```

Figure 82. Sample WebSphere MQ for Linux trace

Figure 83 on page 3013 shows an extract from a WebSphere MQ for AIX trace:

```
Timestamp        Process.Thread  Trace Ident  Trace Data
=========================================================
 12:06:32.904335   622742.1           :           Header.v02:7.0:AIX 5.3:64:-1:1:GMT
 12:06:32.904427   622742.1           :           Version : 7.0.0.0    Level : p000-L090514
 12:06:32.904540   622742.1           :           UTC   Date : 05/15/09  Time : 11:06:32.904302
 12:06:32.904594   622742.1           :           Local Date : 05/15/09  Time : 12:06:32.904302  GMT
 12:06:32.904697   622742.1           :           PID : 622742 Process : dltmqm_nd (64-bit)
 12:06:32.904728   622742.1           :           Host : dynamo
 12:06:32.904755   622742.1           :           Operating System : AIX 5.3
 12:06:32.904781   622742.1           :           Product Long Name : WebSphere MQ for AIX
 12:06:32.904806   622742.1           :           --------------------------------
 12:06:32.904832   622742.1           :           xtrNullFd: 3, xihTraceFileNum: 5
 12:06:32.904916   622742.1           :           Data: 0x00000000
 12:06:32.904952   622742.1           :           Thread stack
 12:06:32.904982   622742.1           :           -> InitProcessInitialisation
 12:06:32.905007   622742.1           :      {  InitProcessInitialisation
 12:06:32.905033   622742.1           :      -{  xcsIsEnvironment
 12:06:32.905062   622742.1           :           xcsIsEnvironment[AMQ_NO_CS_RELOAD] = FALSE
 12:06:32.905088   622742.1           :      -}  xcsIsEnvironment rc=OK
 12:06:32.905117   622742.1           :      -{  xcsLoadFunction
 12:06:32.905145   622742.1           :           LibName(libmqmcs_r.a(shr.o)) LoadType(2097200)
 12:06:32.905178   622742.1           :           General, comms, CS, OAM, or WAS
 12:06:32.905204   622742.1           :      --{  xcsQueryValueForSubpool
 12:06:32.905282   622742.1           :      --}  xcsQueryValueForSubpool rc=OK
 12:06:32.905504   622742.1           :           FullPathLibName(/usr/mqm/lib64/libmqmcs_r.a(shr.o)) loaded with load
 12:06:32.905540   622742.1           :      --{  xcsGetMem
 12:06:32.905575   622742.1           :           component:24 function:176 length:2088 options:0 cbmindex:-1 *pointer:
 12:06:32.905601   622742.1           :      --}  xcsGetMem rc=OK
 12:06:32.905638   622742.1           :           Handle(0) Function(0) FullPathLibName(/usr/mqm/lib64/libmqmcs_r.a(shr
 12:06:32.905665   622742.1           :      -}  xcsLoadFunction rc=OK
```

Figure 83. Sample WebSphere MQ for AIX trace

Examples of trace output

Use this topic as an example of how to interpret trace output.

Figure 84 on page 3014 shows an example of a trace taken on entry to an MQPUT1 call. The following items have been produced:

- Queue request parameter list
- Object descriptor (MQOD)
- Message descriptor (MQMD)
- Put message options (MQPMO)
- The first 256 bytes of message data

Compare this to Figure 85 on page 3015, which illustrates the same control blocks on exit from WebSphere MQ.

```
USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
PUTONE
    Thread... 004C2B10  Userid... CICSUSER  pObjDesc. 106B2010
    pMsgDesc. 106B20B8  pPMO..... 106B2200
    BufferL.. 00000064  pBuffer.. 106A0578  RSV1..... 00000000
    RSV2..... 00000000  RSV3..... 116BC830
    C9E8C1E8  C5C3C9C3  AA8E8583  76270484  | IYAYECIC..ec...d |
    D4D8E3E3  0000048C  00000000  00000000  | MQTT............ |
    00000000  1910C7C2  C9C2D4C9  E8C14BC9  | ......GBIBMIYA.I |
    C7C3E2F2  F0F48E85  83762979  00010000  | GCS204.ec..`.... |

               GMT-01/30/05 14:42:08.412320    LOC-01/30/05 14:42:08.412320

USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
 +0000  D6C44040  00000001  00000000  C2404040  | OD  ........B    |
 +0010  40404040  40404040  40404040  40404040  |                  |
 ...
 +00A0  00000000  00000000                     | ........         |

               GMT-01/30/05 14:42:08.412345    LOC-01/30/05 14:42:08.412345

USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
 +0000  D4C44040  00000001  00000000  00000008  | MD  ............ |
 ...
 +0130  40404040  40404040  40404040  40404040  |                  |
 +0140  40404040                               |                  |

               GMT-01/30/05 14:42:08.412370    LOC-01/30/05 14:42:08.412370

USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
 +0000  D7D4D640  00000001  00000000  FFFFFFFF  | PMO ............ |
 ...
 +0070  40404040  40404040  40404040  40404040  |                  |

               GMT-01/30/05 14:42:08.412393    LOC-01/30/05 14:42:08.412393

USRD9 5E9 ASCB 00F87E80           JOBN ECIC330
CSQW072I ENTRY: MQ user parameter trace
 +0000  C1C1C1C1  C1C1C1C1  C1404040  40404040  | AAAAAAAAA        |
 ...
 +0060  40404040                               |                  |

               GMT-01/30/05 14:42:08.412625    LOC-01/30/05 14:42:08.412625
```

Figure 84. Example trace data from an entry trace of an MQPUT1 request

```
USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
PUTONE
    Thread... 004C2B10  Userid... CICSUSER  pObjDesc. 106B2010
    pMsgDesc. 106B20B8  pPMO..... 106B2200
    BufferL.. 00000064  pBuffer.. 106A0578  RSV1..... 00000000
    RSV2..... 00000000  RSV3..... 116BC830
    CompCode. 00000002  Reason... 000007FB
    C9E8C1E8  C5C3C9C3  AA8E8583  76270484  | IYAYECIC..ec...d |
    D4D8E3E3  0000048C  00000000  00000000  | MQTT............ |
    00000000  1910C7C2  C9C2D4C9  E8C14BC9  | ......GBIBMIYA.I |
    C7C3E2F2  F0F48E85  83762979  00010000  | GCS204.ec..`.... |
MQRC_OBJECT_TYPE_ERROR

              GMT-01/30/05 14:42:08.412678    LOC-01/30/05 14:42:08.412678

USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  D6C44040  00000001  00000000  C2404040  | OD  ........B    |
 ...
 +00A0  00000000  00000000                     | ........        |

              GMT-01/30/05 14:42:08.412789    LOC-01/30/05 14:42:08.412789

USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  D4C44040  00000001  00000000  00000008  | MD  ............ |
 ...
 +0140  40404040                               |                 |

              GMT-01/30/05 14:42:08.412814    LOC-01/30/05 14:42:08.412814

USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  D7D4D640  00000001  00000000  FFFFFFFF  | PMO ............ |
 ...
 +0070  40404040  40404040  40404040  40404040  |                 |

              GMT-01/30/05 14:42:08.412836    LOC-01/30/05 14:42:08.412836

USRD9 5EA ASCB 00F87E80           JOBN ECIC330
CSQW073I EXIT:  MQ user parameter trace
 +0000  C1C1C1C1  C1C1C1C1  C1404040  40404040  | AAAAAAAAA        |
 ...
 +0060  40404040                               |                 |

              GMT-01/30/05 14:42:08.412858    LOC-01/30/05 14:42:08.412858
```

Figure 85. Example trace data from an exit trace of an MQPUT1 request

Examples of CEDF output

Use this topic as a reference for example CEDF output from MQI calls.

This topic gives examples of the output produced by the CICS execution diagnostic facility (CEDF) when using WebSphere MQ. The examples show the data produced on entry to and exit from the following MQI calls, in both hexadecimal and character format. Other MQI calls produce similar data.

Example CEDF output for the MQOPEN call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object descriptor
ARG 002	Options
ARG 003	Object handle
ARG 004	Completion code
ARG 005	Reason code

```
  STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 001 (X'D6C4404000000010000000 1C3C5C4C6')            AT X'00144910'
 001: ARG 002 (X'00000072000000000000000000000000')            AT X'001445E8'
 001: ARG 003 (X'00000000000000720000000000000000')            AT X'001445E4'
 001: ARG 004 (X'00000000000000000000000000000000')            AT X'001445EC'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'001445F0'
```

Figure 86. Example CEDF output on entry to an MQOPEN call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 001 (X'D6C4404000000010000000 1C3C5C4C6')            AT X'00144910'
 001: ARG 002 (X'00000072000000000000000000000000')            AT X'001445E8'
 001: ARG 003 (X'00000001000000720000000000000000')            AT X'001445E4'
 001: ARG 004 (X'00000000000000000000000000000000')            AT X'001445EC'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'001445F0'
```

Figure 87. Example CEDF output on exit from an MQOPEN call (hexadecimal)

```
  STATUS:  ABOUT TO EXECUTE COMMAND
  CALL TO RESOURCE MANAGER MQM
   001: ARG 000 ('.............. .')
   001: ARG 001 ('OD  ........CEDF')
   001: ARG 002 ('................')
   001: ARG 003 ('................')
   001: ARG 004 ('................')
   001: ARG 005 ('................')
```

Figure 88. Example CEDF output on entry to an MQOPEN call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('.............. .')
 001: ARG 001 ('OD  ........CEDF')
 001: ARG 002 ('................')
 001: ARG 003 ('................')
 001: ARG 004 ('................')
 001: ARG 005 ('................')
```

Figure 89. Example CEDF output on exit from an MQOPEN call (character)

Example CEDF output for the MQCLOSE call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Options
ARG 003	Completion code
ARG 004	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 003 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 004 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 90. Example CEDF output on entry to an MQCLOSE call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000000000007200000000')            AT X'001445E0'
 001: ARG 001 (X'00000000000000720000000000000000')            AT X'001445E4'
 001: ARG 002 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 003 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 004 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 91. Example CEDF output on exit from an MQCLOSE call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('.............. .')
 001: ARG 003 ('................')
 001: ARG 004 ('...............-')
```

Figure 92. Example CEDF output on entry to an MQCLOSE call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('.............. .')
 001: ARG 003 ('................')
 001: ARG 004 ('...............-')
```

Figure 93. Example CEDF output on exit from an MQCLOSE call (character)

Example CEDF output for the MQPUT call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Message descriptor
ARG 003	Put message options
ARG 004	Buffer length
ARG 005	Message data
ARG 006	Completion code
ARG 007	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'00000001000000720000000000000000')            AT X'001445E4'
 001: ARG 002 (X'D4C4404000000001000000000000008')             AT X'001449B8'
 001: ARG 003 (X'D7D4D64000000001000000240000000')             AT X'00144B48'
 001: ARG 004 (X'00000008000000000000000000040000')            AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')            AT X'00144BF8'
 001: ARG 006 (X'00000000000000000000000800000000')            AT X'001445EC'
 001: ARG 007 (X'00000000000000080000000000000000')            AT X'001445F0'
```

Figure 94. Example CEDF output on entry to an MQPUT call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000000')          AT X'001445E4'
 001: ARG 002 (X'D4C4404000000001000000000000000008')          AT X'001449B8'
 001: ARG 003 (X'D7D4D64000000001000000002400000000')          AT X'00144B48'
 001: ARG 004 (X'00000008000000000000000000040000')            AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')            AT X'00144BF8'
 001: ARG 006 (X'00000000000000000000000800000000')            AT X'001445EC'
 001: ARG 007 (X'00000000000000080000000000000000')            AT X'001445F0'
```

Figure 95. Example CEDF output on exit from an MQPUT call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('PMO .............')
 001: ARG 004 ('................')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('................')
```

Figure 96. Example CEDF output on entry to an MQPUT call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('PMO .............')
 001: ARG 004 ('................')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('................')
```

Figure 97. Example CEDF output on exit from an MQPUT call (character)

Example CEDF output for the MQPUT1 call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object descriptor
ARG 002	Message descriptor
ARG 003	Put message options
ARG 004	Buffer length
ARG 005	Message data
ARG 006	Completion code
ARG 007	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000000000007200000000')            AT X'001445E0'
 001: ARG 001 (X'D6C4404000000010000000 1C3C5C4C6')            AT X'00144910'
 001: ARG 002 (X'D4C4404000000010000000000000008')             AT X'001449B8'
 001: ARG 003 (X'D7D4D64000000010000002400000000')             AT X'00144B48'
 001: ARG 004 (X'00000008000000080000006000040000')            AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')            AT X'00144BF8'
 001: ARG 006 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 007 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 98. Example CEDF output on entry to an MQPUT1 call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000000000007200000000')            AT X'001445E0'
 001: ARG 001 (X'D6C4404000000010000000 1C3C5C4C6')            AT X'00144910'
 001: ARG 002 (X'D4C4404000000010000000000000008')             AT X'001449B8'
 001: ARG 003 (X'D7D4D64000000010000002400000000')             AT X'00144B48'
 001: ARG 004 (X'00000008000000080000006000040000')            AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')            AT X'00144BF8'
 001: ARG 006 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 007 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 99. Example CEDF output on exit from an MQPUT1 call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('OD  ........CEDF')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('PMO ............')
 001: ARG 004 ('...........-....')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('...............-')
```

Figure 100. Example CEDF output on entry to an MQPUT1 call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('OD  ........CEDF')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('PMO ............')
 001: ARG 004 ('...........-....')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('...............-')
```

Figure 101. Example CEDF output on exit from an MQPUT1 call (character)

Example CEDF output for the MQGET call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Message descriptor
ARG 003	Get message options
ARG 004	Buffer length
ARG 005	Message buffer
ARG 006	Message length
ARG 007	Completion code
ARG 008	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'D4C4404000000001000000000000000')             AT X'001449B8'
 001: ARG 003 (X'C7D4D6400000000100004044FFFFFFFF')            AT X'00144B00'
 001: ARG 004 (X'00000008000000000000000000040000')            AT X'001445F4'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'00144C00'
 001: ARG 006 (X'00000000000000000004000000000000')            AT X'001445F8'
 001: ARG 007 (X'00000000000000000000000800000000')            AT X'001445EC'
 001: ARG 008 (X'00000000000000080000000000000000')            AT X'001445F0'
```

Figure 102. Example CEDF output on entry to an MQGET call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'D4C4404000000001000000000000008')             AT X'001449B8'
 001: ARG 003 (X'C7D4D6400000000100004044FFFFFFFF')            AT X'00144B00'
 001: ARG 004 (X'00000008000000080000000000040000')            AT X'001445F4'
 001: ARG 005 (X'5C5CC8C5D3D3D640E6D6D9D3C45C5C5C')            AT X'00144C00'
 001: ARG 006 (X'00000008000000000004000000000000')            AT X'001445F8'
 001: ARG 007 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 008 (X'00000000000000080000000800000000')            AT X'001445F0'
```

Figure 103. Example CEDF output on exit from an MQGET call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('GMO ...... .....')
 001: ARG 004 ('................')
 001: ARG 005 ('................')
 001: ARG 006 ('................')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
```

Figure 104. Example CEDF output on entry to an MQGET call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('MD  ............')
 001: ARG 003 ('GMO ...... .....')
 001: ARG 004 ('................')
 001: ARG 005 ('**HELLO WORLD***')
 001: ARG 006 ('................')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
```

Figure 105. Example CEDF output on exit from an MQGET call (character)

Example CEDF output for the MQINQ call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Count of selectors
ARG 003	Array of attribute selectors
ARG 004	Count of integer attributes
ARG 005	Integer attributes
ARG 006	Length of character attributes buffer
ARG 007	Character attributes
ARG 008	Completion code
ARG 009	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000000200004044')            AT X'05ECAFCC'
 001: ARG 001 (X'0000000100000072000000000000000'0')           AT X'001445E4'
 001: ARG 002 (X'0000000200004044 85ECA00885ECA220')           AT X'05ECAFD4'
 001: ARG 003 (X'0000000D0000000C0000000000000000')            AT X'00144C08'
 001: ARG 004 (X'0000000200004044 85ECA00885ECA220')           AT X'05ECAFD4'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'00144C10'
 001: ARG 006 (X'00000000000000010000000200004044')            AT X'05ECAFCC'
 001: ARG 007 (X'00000000000000000000000000000000')            AT X'00144C18'
 001: ARG 008 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 009 (X'00000000000000080000000800040000')            AT X'001445F0'
```

Figure 106. Example CEDF output on entry to an MQINQ call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000000200004044')            AT X'05ECAFCC'
 001: ARG 001 (X'0000000100000072000000000000000'0')           AT X'001445E4'
 001: ARG 002 (X'0000000200004044 85ECA00885ECA220')           AT X'05ECAFD4'
 001: ARG 003 (X'0000000D0000000C0040000000000000')            AT X'00144C08'
 001: ARG 004 (X'0000000200004044 85ECA00885ECA220')           AT X'05ECAFD4'
 001: ARG 005 (X'00400000000000000000000000000000')            AT X'00144C10'
 001: ARG 006 (X'00000000000000010000000200004044')            AT X'05ECAFCC'
 001: ARG 007 (X'00000000000000000000000000000000')            AT X'00144C18'
 001: ARG 008 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 009 (X'00000000000000080000000800040000')            AT X'001445F0'
```

Figure 107. Example CEDF output on exit from an MQINQ call (hexadecimal)

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('.............. .')
 001: ARG 001 ('................')
 001: ARG 002 ('...... .e...e.s.')
 001: ARG 003 ('................')
 001: ARG 004 ('...... .e...e.s.')
 001: ARG 005 ('................')
 001: ARG 006 ('.............. .')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
 001: ARG 009 ('................')
```

Figure 108. Example CEDF output on entry to an MQINQ call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('.............. .')
 001: ARG 001 ('................')
 001: ARG 002 ('...... .e...e.s.')
 001: ARG 003 ('......... ......')
 001: ARG 004 ('...... .e...e.s.')
 001: ARG 005 ('. ..............')
 001: ARG 006 ('.............. .')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
 001: ARG 009 ('................')
```

Figure 109. Example CEDF output on exit from an MQINQ call (character)

Example CEDF output for the MQSET call

The parameters for this call are:

Parameter	Description
ARG 000	Connection handle
ARG 001	Object handle
ARG 002	Count of selectors
ARG 003	Array of attribute selectors
ARG 004	Count of integer attributes
ARG 005	Integer attributes
ARG 006	Length of character attributes buffer
ARG 007	Character attributes
ARG 008	Completion code
ARG 009	Reason code

```
STATUS:  ABOUT TO EXECUTE COMMAND
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000000')          AT X'001445E4'
 001: ARG 002 (X'0000000100000002000040485ECA008')             AT X'05ECAFDC'
 001: ARG 003 (X'00000018000007DF0000000000000000')            AT X'00144C08'
 001: ARG 004 (X'0000000100000002000040485ECA008')             AT X'05ECAFDC'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'00144C10'
 001: ARG 006 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 007 (X'00000000000000000000000000000000')            AT X'00144C18'
 001: ARG 008 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 009 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 110. Example CEDF output on entry to an MQSET call (hexadecimal)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 (X'00000000000000010000007200000000')            AT X'001445E0'
 001: ARG 001 (X'0000000100000072000000000000000')             AT X'001445E4'
 001: ARG 002 (X'0000000100000002000040448 5ECA008')           AT X'05ECAFDC'
 001: ARG 003 (X'00000018000007DF0000000000000000')            AT X'00144C08'
 001: ARG 004 (X'0000000100000002000040448 5ECA008')           AT X'05ECAFDC'
 001: ARG 005 (X'00000000000000000000000000000000')            AT X'00144C10'
 001: ARG 006 (X'00000000000000010000000200004044')            AT X'05ECAFD8'
 001: ARG 007 (X'00000000000000000000000000000000')            AT X'00144C18'
 001: ARG 008 (X'00000000000000000000000800000008')            AT X'001445EC'
 001: ARG 009 (X'00000000000000080000000800000060')            AT X'001445F0'
```

Figure 111. Example CEDF output on exit from an MQSET call (hexadecimal)

```
 STATUS:  ABOUT TO EXECUTE COMMAND
 CALL TO RESOURCE MANAGER MQM
  001: ARG 000 ('................')
  001: ARG 001 ('................')
  001: ARG 002 ('.......... .e...')
  001: ARG 003 ('................')
  001: ARG 004 ('.......... .e...')
  001: ARG 005 ('................')
  001: ARG 006 ('............... .')
  001: ARG 007 ('................')
  001: ARG 008 ('................')
  001: ARG 009 ('................-')
```

Figure 112. Example CEDF output on entry to an MQSET call (character)

```
STATUS:  COMMAND EXECUTION COMPLETE
CALL TO RESOURCE MANAGER MQM
 001: ARG 000 ('................')
 001: ARG 001 ('................')
 001: ARG 002 ('.......... .e...')
 001: ARG 003 ('................')
 001: ARG 004 ('.......... .e...')
 001: ARG 005 ('................')
 001: ARG 006 ('............... .')
 001: ARG 007 ('................')
 001: ARG 008 ('................')
 001: ARG 009 ('................-')
```

Figure 113. Example CEDF output on exit from an MQSET call (character)

Messages

You can use the following messages to help you solve problems with your WebSphere MQ components or applications.

Diagnostic messages: AMQ4000-9999

Diagnostic messages are listed in this section in numeric order, grouped according to the part of WebSphere MQ from which they originate.

- AMQ4000-4999: User interface messages (WebSphere MQ for Windows and Linux systems)
- AMQ5000-5999: Installable services
- AMQ6000-6999: Common services
- AMQ7000-7999: WebSphere MQ
- AMQ8000-8999: Administration
- AMQ9000-9999: Remote

Reading a message

For each message, this information is provided:

- The message identifier, in two parts:
  1. The characters "AMQ" which identify the message as being from WebSphere MQ
  2. A four-digit decimal code

  If a message is specific to a single platform, this is indicated after the message identifier. Although some messages are listed several times, each instance relates to a different platform. If present, the version common to a number of platforms is listed first, followed by versions for individual platforms. Ensure that you read the appropriate version.
- The text of the message.
- The severity of the message:

  0: Information
  10: Warning
  20: Error
  30: Severe error
  40: Stop Error
  50: System Error
- An explanation of the message giving further information.
- The response required from the user. In some cases, particularly for information messages, this might be "none".

Message variables

Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as *message variables*. The message variables are indicated as <insert_1>, <insert_2>, and so on.

In some cases a message might have variables in the Explanation or Response. Find the values of the message variables by looking in the error log. The complete message, including the Explanation and the Response, is recorded there.

Related information:

API completion and reason codes

PCF reason codes

Secure Sockets Layer (SSL) return codes

WCF custom channel exceptions

AMQ4000-4999: User interface messages (WebSphere MQ for Windows and Linux systems)

AMQ4000

New object not created because the default object for the object type could not be found.

Severity

10: Warning

Explanation

The creation of an object requires a default template for each object type. The required default template for this object type could not be found.

Response

Determine why the default object is unavailable, or create a new one. Then try the request again.

AMQ4001

The queue manager specified has already been added to WebSphere MQ Explorer.

Severity

0: Information

Response

Message for information only. If the queue manager is not displayed in the Navigator view ensure that the queue manager is not hidden.

AMQ4002

Are you sure that you want to delete the object named <*insert_0*>?

Severity

10: Warning

Explanation

A confirmation is required before the specified object is deleted. The type of object and name are provided in the message.

Response

Continue only if you want to permanently delete the object.

AMQ4003

WebSphere MQ system objects are used internally by WebSphere MQ. You are advised not to delete them. Do you want to keep the WebSphere MQ system object?

Severity

0: Information

Explanation

A confirmation is required before an internal WebSphere MQ system object (for example SYSTEM.DEFAULT.LOCAL.QUEUE) is deleted.

Response

Continue only if you want to permanently delete the system object.

AMQ4004

Clear all messages from the queue?

Severity

10: Warning

Explanation
The removal of the messages from the queue is an irreversible action. If the command is allowed to proceed the action cannot be undone.

Response
Continue only if you want to permanently delete the messages.

AMQ4005
The object has been replaced or deleted. The properties could not be applied.

Severity
10: Warning

Explanation
During the process of updating the properties of the object, it was determined that the object has either been deleted or replaced by another instance. The properties have not been applied.

AMQ4006
WebSphere MQ successfully sent data to the remote queue manager and received the data returned.

Severity
0: Information

Explanation
An open channel has been successfully verified by WebSphere MQ as the result of a user request.

Response
Message for information only.

AMQ4007
The message sequence number for the channel was reset.

Severity
0: Information

Explanation
A channel has had its sequence number successfully reset by WebSphere MQ as the result of a user request.

Response
Message for information only.

AMQ4008
The request to start the channel was accepted.

Severity
0: Information

Explanation
A channel has been started successfully by WebSphere MQ as the result of a user request.

Response
Message for information only.

AMQ4009
The request to stop the channel was accepted.

Severity
0: Information

Explanation
A channel has been stopped successfully by WebSphere MQ as the result of a user request.

Response
Message for information only.

AMQ4010
The 'in-doubt' state was resolved.

Severity
0: Information

Explanation
A channel has had its 'in-doubt' state resolved successfully by WebSphere MQ as the result of a user request.

Response
Message for information only

AMQ4011
The queue has been cleared of messages.

Severity
0: Information

Explanation
The CLEAR command has completed successfully and has removed all messages from the target queue. If the CLEAR was performed using the MQGET API command, uncommitted messages might still be on the queue.

AMQ4012
The object was created successfully but it is not visible with the current settings for visible objects.

Severity
0: Information

Response
Message for information only.

AMQ4014
The character <*insert_0*> is not valid.

Severity
10: Warning

AMQ4015
Supply a non-blank name.

Severity
0: Information

Response
Enter a valid name.

AMQ4016
The test message was put successfully.

Severity
0: Information

Explanation
The request to place a message on the target queue has completed successfully. The queue now contains the message.

Response
Message for information only.

AMQ4019
An object called <*insert_0*> exists. Do you want to replace the definition of the existing object?

Severity
0: Information

Response
Confirm that you want to replace the definition.

AMQ4020
The changes you are making to the attributes of page <*insert_0*> will affect the operation of the queue manager or another program currently using the object. Do you want to force the change to the object's attributes?

Severity
10: Warning

Explanation
You are trying to change an object that cannot be changed because it is in use, or the change affects other programs or queue managers. Some changes can be forced anyway.

Response
Select Yes to try forcing the changes, or No to abandon the change.

AMQ4021
Failed to access one or more WebSphere MQ objects.

Severity
10: Warning

Explanation
The icons of the objects have been marked to indicate the objects in error.

AMQ4022
The name specified for the initiation queue is the same as the name of the queue itself.

Severity
0: Information

Response
Specify a different name to that of the object being created or altered.

AMQ4023
The queue manager <*insert_0*> does not exist on this computer.

Severity
0: Information

Response
Message for information only.

AMQ4024
The object cannot be replaced.

Severity
0: Information

Explanation
The request to replace the object was unsuccessful.

Response
To define this object, delete the existing object and try the operation again.

AMQ4025
The changes made to the cluster attributes of the queue take effect once they have propagated across the network.

Severity
0: Information

Response
Refresh any views containing the cluster queues in the affected clusters to show the changes.

AMQ4026
You have created a queue which is shared in one or more clusters. The queue will be available as a cluster queue once its definition has propagated across the network.

Severity
0: Information

Response
Refresh any views containing the cluster queues in the affected clusters to show the cluster queue.

AMQ4027
An error occurred connecting to queue manager <*insert_0*>. Are you sure that you want to show this queue manager in the folder anyway?

Severity
10: Warning

Explanation
A connection could not be made to the specified remote queue manager.

Response
Ensure that the named queue manager is running on the host and port specified, and has a channel corresponding to the specified name. Ensure that you have the authority to connect to the remote queue manager, and ensure that the network is running. Select Yes if you believe that the problem can be resolved later. Select No if you want to correct the problem now and try again.

AMQ4028
Platform not supported. This queue manager cannot be administered by the WebSphere MQ Explorer because it is running on an unsupported platform. The value <*insert_0*> for the Platform attribute of the queue manager is not supported by the WebSphere MQ Explorer.

Severity
20: Error

AMQ4029
Command level too low. This queue manager cannot be administered by the WebSphere MQ Explorer.

Severity
20: Error

Response
If you want to administer this queue manager, you must upgrade it to a newer version of WebSphere MQ.

AMQ4030
Queue manager cannot be administered because code page conversion table not found.

Severity
20: Error

Explanation
This queue manager cannot be administered by the WebSphere MQ Explorer because a code page conversion table was not found.

Response
Install a code page conversion table from CCSID <*insert_0*> to CCSID <*insert_1*> on the computer on which the WebSphere MQ Explorer is running.

AMQ4031
Queue manager cannot be administered because CCSID not found.

Severity
20: Error

Explanation
This queue manager cannot be administered by the WebSphere MQ Explorer because CCSID <*insert_0*> cannot be found in the CCSID table. The WebSphere MQ Explorer cannot convert character data to or from the unrecognized CCSID.

AMQ4032
Command server not responding within timeout period.

Severity
10: Warning

Response
Ensure that the command server is running and that the queue called 'SYSTEM.ADMIN.COMMAND.QUEUE' is configured to enable programs to get messages from it.

AMQ4033
Cannot get messages from the queue.

Severity
0: Information

Explanation
A reason code returned when the object was opened for input indicated that the queue is disabled for MQGET request.

Response
To get messages from this queue, enable it for GET requests.

AMQ4034
Message too long. You tried to put a message on a queue that was bigger than the maximum allowed for the queue or queue manager.

Severity
10: Warning

Explanation
The request to put a message on a queue returned a reason code indicating that the data length of the message exceeds the maximum allowed in the definition of the queue.

Response
Either change the MAXMSGL attribute of the queue so that it is equal to or greater than the length of the message, or reduce the length of the message being put on the queue.

AMQ4035
No message available. The response message did not arrive within a reasonable amount of time.

Severity
0: Information

Explanation
The request to get a message from a queue returned a reason code indicating that there are currently no messages on the queue that meet the selection criteria specified on the GET request.

AMQ4036
Access not permitted. You are not authorized to perform this operation.

Severity
10: Warning

Explanation

The security mechanism of the queue manager has indicated that the user ID associated with this request is not authorized to access the object.

AMQ4037

Object definition changed since it was opened.

Severity

0: Information

Explanation

Object definitions that affect this object have been changed since the Hobj handle used on this call was returned by the MQOPEN call.

Response

Issue an MQCLOSE call to return the handle to the system. It is then normally sufficient to reopen the object and try the operation again.

AMQ4038

Object damaged.

Severity

10: Warning

Explanation

The object is damaged and cannot be accessed.

Response

The object must be deleted. Alternatively, it might be possible to recover it from a media image or backup.

AMQ4039

Object in use. The object is already opened by another application.

Severity

10: Warning

Explanation

An MQOPEN call was issued, but the object in question has already been opened by this application or another application with options that conflict with those options specified in the Options parameter. This situation arises if the request is for shared input, but the object is already open for exclusive input. It also arises if the request is for exclusive input, but the object is already open for input (of any sort).

Response

To change the attributes of an object, specify the Force option as 'Yes' to apply the changes. If you specify the Force option as 'Yes', any applications using the object must close and reopen the object to proceed.

AMQ4040

Cannot put messages on this queue.

Severity

0: Information

Explanation

MQPUT and MQPUT1 calls are currently inhibited for the queue, or for the queue to which this queue resolves.

AMQ4042

Queue full. The queue contains the maximum number of messages.

Severity

10: Warning

Explanation
On an MQPUT or MQPUT1 call, the call failed because the queue is full; that is, it already contains the maximum number of messages possible.

AMQ4043
Queue manager not available for connection.

Severity
20: Error

Response
Ensure that the queue manager is running. If the queue manager is running on another computer, ensure that it is configured to accept remote connections.

AMQ4044
Queue manager <*insert_0*> is stopping.

Severity
0: Information

Explanation
An MQI call was issued, but the call failed because the queue manager is shutting down. If the call was an MQGET call with the MQGMO_WAIT option, the wait has been canceled.

Response
You cannot issue any more MQI calls.

AMQ4045
Queue not empty. The queue contains one or more messages or uncommitted PUT or GET requests.

Severity
0: Information

Explanation
An operation that requires the queue to be empty has failed because the queue either contains messages or has uncommitted PUT or GET requests outstanding.

AMQ4046
Insufficient system resources available.

Severity
20: Error

AMQ4047
Insufficient storage available.

Severity
20: Error

AMQ4048
The request received an unexpected reason code from an underlying API or command request. The reason code was <*insert_0*>.

Severity
20: Error

Explanation
While running the requested operation, an unexpected return code was received, resulting in the operation not completing as expected.

Response
Use the reason code to determine the underlying reason for the failure.

AMQ4049
Unknown object name.

Severity
10: Warning

Explanation
A command or API request was issued, but the object cannot be found.

AMQ4050
Allocation failed. An attempt to allocate a conversation to a remote system failed.

Severity
10: Warning

Explanation
The error might be due to an incorrect entry in the channel definition or it might be that the listening program on the remote system was not running.

AMQ4051
Bind failed. The bind to a remote system during session negotiation failed.

Severity
10: Warning

AMQ4052
Coded character-set ID error. Cannot convert a command message to the CCSID of the target queue manager.

Severity
10: Warning

AMQ4053
Channel in doubt. Operation not completed.

Severity
10: Warning

Explanation
The operation could not be completed because the channel was in doubt.

AMQ4054
Channel in use.

Severity
10: Warning

Explanation
An attempt was made to perform an operation on a channel, but the channel is currently active.

AMQ4055
Channel status not found.

Severity
10: Warning

Explanation
No channel status is available for this channel, possibly indicating that the channel has not been used.

AMQ4056
Command failed.

Severity
10: Warning

AMQ4057
Configuration error in the channel definition or communication subsystem.

Severity
10: Warning

Explanation
Allocation of a conversation is not possible.

AMQ4058
Connection closed.

Severity
10: Warning

Explanation
The connection to a remote system has unexpectedly broken while receiving data.

AMQ4059
Could not establish a connection to the queue manager.

Severity
10: Warning

Explanation
The attempt to connect to the queue manager failed. This failure might be because the queue manager is incorrectly configured to allow a connection from this system, or the connection has been broken.

Response
Try the operation again. If the error persists, examine the problem determination information to see if any information has been recorded.

AMQ4060
Dynamic queue scope error.

Severity
10: Warning

Explanation
The Scope attribute of the queue was set to MQSCO_CELL but this value is not allowed for a dynamic queue.

AMQ4061
Remote system unavailable and unable to allocate a conversation to a remote system.

Severity
10: Warning

Response
The error might be transitory; try again later.

AMQ4062
An MQINQ call failed when the queue manager inquired about a WebSphere MQ object.

Severity
10: Warning

Response
Check the error log of the queue manager for more information about the error.

AMQ4063
An MQOPEN call failed when the queue manager tried to open a WebSphere MQ object.

Severity
20: Error

Response

If the error occurred while starting a channel check that the transmission queue used by the channel exists and try the operation again. If the error persists check the error log of the queue manager for more information about the error.

AMQ4064

An MQSET call failed when the queue manager tried to set the values of the attributes of a WebSphere MQ object.

Severity

10: Warning

Response

Check the error log of the queue manager for more information about the error.

AMQ4065

Message sequence number error.

Severity

10: Warning

Explanation

The message sequence number parameter was not valid.

AMQ4066

Message truncated because it is larger than the command server's maximum valid message size.

Severity

10: Warning

AMQ4067

Communications manager not available.

Severity

20: Error

Explanation

The communications subsystem is unavailable.

AMQ4068

The queue specified in the channel definition is not a transmission queue, or is in use.

Severity

10: Warning

AMQ4069

Object already exists.

Severity

10: Warning

Explanation

Unable to create object because the object already existed.

AMQ4070

Object is open.

Severity

10: Warning

Explanation

An attempt was made to delete, change, or clear an object that is in use.

Response

Wait until the object is not in use, then try again.

AMQ4071
Object has wrong type. Could not replace a queue object of a different type.

Severity
10: Warning

AMQ4072
Queue already exists in cell.

Severity
10: Warning

Explanation
cannot define a queue with cell scope or change the scope of an existing queue from queue-manager scope to cell scope, because a queue with that name already exists in the cell.

AMQ4073
Ping error. You can only ping a sender or server channel. If the local channel is a receiver channel, ping from the remote queue manager.

Severity
10: Warning

AMQ4074
Receive failed, possibly due to a communications failure.

Severity
10: Warning

AMQ4075
Error while receiving data from a remote system, possibly due to a communications failure.

Severity
10: Warning

AMQ4076
Remote queue manager terminating.

Severity
10: Warning

Explanation
The channel stopped because the remote queue manager was terminating.

AMQ4077
Remote queue manager not available.

Severity
10: Warning

Explanation
The channel could not be started because the remote queue manager was not available.

Response
Ensure that the remote queue manager is started, and that it is configured to accept incoming communication requests.

AMQ4078
Send failed. An error occurred while sending data to a remote system, possibly due to a communications failure.

Severity
10: Warning

AMQ4079
Channel closed by security exit.

Severity
10: Warning

AMQ4080
Remote channel not known.

Severity
10: Warning

Explanation
There is no definition of this channel on the remote system.

AMQ4081
User exit not available.

Severity
10: Warning

Explanation
The channel was closed because the user exit specified does not exist.

AMQ4082
Unexpected WebSphere MQ error (<*insert_0*>).

Severity
20: Error

AMQ4083
Queue manager name not known.

Severity
10: Warning

Explanation
If the queue manager is remote, this might indicate that another queue manager is incorrectly using the same connection name. Queue managers using TCP/IP on the same computer must listen on different port numbers. This means that they will also have different connection names.

AMQ4084
Cell directory is not available.

Severity
10: Warning

Explanation
The Scope attribute of the queue was set to MQSCO_CELL but no name service supporting a cell directory has been configured.

Response
Configure a name service to support the cell directory.

AMQ4085
No name supplied for transmission queue.

Severity
10: Warning

Response
Supply a non-blank transmission queue name for this channel type.

AMQ4086
No connection name supplied.

Severity
10: Warning

Response
Supply a non-blank connection name for this channel type.

AMQ4087
An error occurred while trying to use a cluster resource.

Severity
10: Warning

Response
Check that the queues with names that start with 'SYSTEM.CLUSTER.' are not full and that messages are allowed to be put on them.

AMQ4088
Cannot share transmission queue in cluster.

Severity
10: Warning

Explanation
The queue is a transmission queue and cannot be shared in a cluster.

AMQ4089
PUT commands inhibited for system command queue called <*insert_0*>.

Severity
10: Warning

AMQ4090
Are you sure that you want to inhibit PUT and GET commands for the queue called 'SYSTEM.ADMIN.COMMAND.QUEUE'? If you do, you will no longer be able to administer the queue manager using the WebSphere MQ Explorer.

Severity
10: Warning

Explanation
WebSphere MQ Explorer uses the queue called 'SYSTEM.ADMIN.COMMAND.QUEUE' to administer the queue manager.

Response
Continue only if you really want to inhibit PUT or GET commands for this queue and stop using the WebSphere MQ Explorer to administer the queue manager.

AMQ4091
Cannot connect to remote queue manager.

Severity
10: Warning

Explanation
The remote queue manager is using an unsupported protocol for connections. The WebSphere MQ Explorer only supports connections to remote queue managers using the TCP/IP protocol.

AMQ4092
The queue manager could not be removed from the cluster because its membership of the cluster is defined using namelist <*insert_0*>.

Severity
10: Warning

Response
To remove the queue manager from the cluster, remove it from the namelist. Ensure that you do not inadvertently affect the definitions of other objects using the namelist.

AMQ4093
The cluster specified is already shown in the console.

Severity
0: Information

AMQ4094
An error occurred adding this cluster to the console. Are you sure that you want to show this cluster in the console anyway?

Severity
10: Warning

Response
Select Yes if you believe that the problem can be resolved later. Select No if you want to correct the problem now and try again.

AMQ4095
Queue manager <*insert_0*> is not a repository queue manager for cluster <*insert_1*>.

Severity
0: Information

Explanation
To administer a cluster, the WebSphere MQ Explorer needs a connection to the repository queue manager.

AMQ4096
Are you sure that you want to clear the password?

Severity
0: Information

Response
Check with the user before clearing the password. Continue only if you really want to clear the password.

AMQ4097
Unmatched quotation mark.

Severity
10: Warning

Explanation
An unmatched quotation mark has been found in a list of attributes. Each value in the list can be enclosed in a pair of single or double quotation marks. (Only required for values which contain spaces, commas, or quotation marks.)

Response
Check that all opening and closing quotation marks are in pairs. (To include a quotation mark within an attribute, use two together with no space between.)

AMQ4098
Incorrect list format.

Severity
10: Warning

Explanation
The attribute can contain a list of values which must be separated by a space or a comma. Each value in the list can be enclosed in a pair of single or double quotation marks. (Only required for values which contain spaces, commas, or quotation marks.)

Response

Check that values are separated by a space or a comma, and that all opening and closing quotation marks are in pairs. (To include a quotation mark within an attribute, use two together with no space between.)

AMQ4099

Cannot communicate with one or more repository queue managers. Cluster <*insert_0*> is configured to use one or more repository queue managers which communicate using a protocol other than TCP/IP.

Severity

10: Warning

Explanation

The WebSphere MQ Explorer can only establish connections to remote queue managers using TCP/IP.

Response

To complete removal of the queue manager from the cluster, issue the RESET CLUSTER ACTION(FORCEREMOVE) command from the repository queue manager.

AMQ4103

An error occurred connecting to the queue manager. Are you sure that you want to show this queue manager in the folder?

Severity

10: Warning

Explanation

A connection could not be made to the specified remote queue manager.

Response

Ensure that the named queue manager is running on the machine specified in the selected channel definition table. Ensure that you have the authority to connect to the remote queue manager, and ensure that the network is operational. Select Yes if you believe that the problem can be resolved later. Select No if you want to correct the problem now and try again.

AMQ4104

The specified file <*insert_0*> does not contain a client definition table in the correct format.

Severity

10: Warning

Explanation

The channel definition table is not in the correct format.

Response

Specify a file in the correct format.

AMQ4105

The remote queue manager has not been removed because it is still required by other plug-ins.

Severity

10: Warning

Explanation

Other plug-ins have responded to the attempted removal of this queue manager by indicating that they are still using it.

Response

Ensure that the other plug-ins have finished using the queue manager before trying to delete it again.

AMQ4117

This action cannot be undone. Are you sure that you want to delete the WebSphere MQ queue manager <*insert_0*> from your system?

Severity

10: Warning

Explanation

A confirmation is required before the queue manager is deleted.

Response

Continue only if you want to permanently delete the queue manager.

AMQ4121

The MQGET request received an unexpected reason code of <*insert_0*>.

Severity

10: Warning

Explanation

An unexpected reason code was returned from an MQGET API request. Use the reason code to determine the underlying reason why the request failed.

Response

The MQGET request was not successful. Some messages might not have been retrieved.

AMQ4122

The MQPUT request received an unexpected reason code of <*insert_0*>.

Severity

10: Warning

Explanation

An unexpected reason code was returned from an MQPUT API request. Use the reason code to determine the underlying reason why the request failed.

Response

MQPUT processing was unsuccessful. No message was placed on the queue.

AMQ4123

The object <*insert_0*> was deleted successfully.

Severity

0: Information

Explanation

The object of the specified name has been successfully deleted.

Response

none.

AMQ4124

The MQOPEN request received an unexpected reason code of <*insert_0*>.

Severity

10: Warning

Explanation

An unexpected reason code was returned from an MQOPEN API request. The queue has not been opened.

Response

Use the reason code to determine the underlying reason for the failure.

AMQ4125

Putting a test message on the queue received an unexpected reason code <*insert_0*>.

Severity
10: Warning

Explanation
One of the underlying API requests was unsuccessful. The test message was not placed on the queue.

AMQ4126
The value of one of the properties specified is not valid. The request was not processed.

Severity
20: Error

Response
Specify a different value.

AMQ4127
WebSphere MQ failed to read queue manager information from disk because the file format is not valid. The request was not processed.

Severity
20: Error

Explanation
The format of the WebSphere MQ_Handles file is incorrect. This file has been backed up and removed, meaning that any remote queue manager definitions are lost. All local queue managers should be detected automatically and displayed in the WebSphere MQ Explorer.

Response
Ensure that the Eclipse workspace has not been corrupted.

AMQ4128
Could not start the iKeyMan program.

Severity
30: Severe error

Explanation
An error was encountered when trying to execute the iKeyMan program.

Response
Try again. If symptoms persist contact your System Administrator.

AMQ4129
Could not query the user ID from Java.

Severity
10: Warning

Explanation
The Java API System.getProperty("user.id") threw a SecurityException.

Response
Configure your Java security environment using the 'policytool' to allow WebSphere MQ Explorer to query the 'user.id'.

AMQ4130
A Browser Control could not be opened. Make sure Mozilla has been installed.

Severity
10: Warning

Explanation
The SWT Browser control depends on Mozilla being installed.

Response
Ensure that the Mozilla browser is correctly installed.

AMQ4131
A Browser Control could not be opened.

Severity
10: Warning

Explanation
The SWT Browser control depends on the system browser being installed.

Response
Ensure that the system browser is correctly installed.

AMQ4132
Are you sure that you want to stop the object named <*insert_0*>?

Severity
10: Warning

Explanation
A confirmation is required before the specified object is stopped. The type of object and name are provided in the message.

Response
Continue only if you want to stop the object.

AMQ4133
When a queue manager is removed, WebSphere MQ Explorer destroys the connection information for that queue manager.

To see the queue manager at a later date use the Add Queue Manager wizard.

Remove the queue manager <*insert_0*> ?

Severity
10: Warning

Response
Continue only if you want to remove the queue manager.

AMQ4134
The default channel used by remote queue managers to administer this queue manager does not exist.

Do you want to create the default remote administration channel SYSTEM.ADMIN.SVRCONN to allow this queue manager to be administered by other queue managers?

Severity
0: Information

Response
Select Yes to create the channel.

AMQ4135
The default channel used by remote queue managers to administer this queue manager is SYSTEM.ADMIN.SVRCONN.

Do you want to delete this channel to prevent the queue manager being administered by other queue managers?

Severity
0: Information

Response
Select Yes to delete the channel.

AMQ4136
This operation deletes all files in the errors and trace directories (including, for example, read only files). This operation cannot be undone. Are you sure that you want to proceed?

Severity
10: Warning

Explanation
Deleting all FFSTs and Trace from this machine means that any historical error logs and trace will be lost.

Response
Select Yes to clear the contents of the errors and trace directories.

AMQ4137
The default remote administration channel SYSTEM.ADMIN.SVRCONN has been deleted successfully.

Severity
0: Information

Response
Message for information only.

AMQ4138
Are you sure that you want to import new settings that will overwrite the current settings? This operation cannot be undone.

Severity
10: Warning

Explanation
Importing settings into the WebSphere MQ Explorer will overwrite the current settings.

Response
Continue only if you want to overwrite the current settings.

AMQ4139
The default remote administration channel SYSTEM.ADMIN.SVRCONN was created successfully.

Severity
0: Information

Response
Message for information only.

AMQ4140
The custom CipherSpec is not valid.

Severity
10: Warning

AMQ4141
The Distinguished Names specification is not valid.

Severity
10: Warning

AMQ4142
The default remote administration channel SYSTEM.ADMIN.SVRCONN could not be created.

Severity
10: Warning

Explanation
A problem has occurred when issuing a command to the command server to create the channel.

Response
Try again. If symptoms persist contact your System Administrator.

AMQ4143
The default remote administration channel SYSTEM.ADMIN.SVRCONN could not be created.

Severity
10: Warning

Explanation
A problem occurred when copying the default administration channel to use as a template for the channel creation.

Response
Try again. If symptoms persist contact your System Administrator.

AMQ4144
The default remote administration channel SYSTEM.ADMIN.SVRCONN could not be deleted.

Severity
10: Warning

Explanation
A problem has occurred issuing a command to the command server to delete the channel.

Response
Ensure that the channel is not in use and try again. If symptoms persist contact your System Administrator.

AMQ4145
An error occurred connecting to the remote queue manager using the intermediate queue manager. Are you sure that you want to show this queue manager in the folder anyway?

Severity
10: Warning

Explanation
A connection could not be made to the specified remote queue manager.

Response
Ensure that the intermediate queue manager is available and that the named remote queue manager is running, and is accessible from the intermediate queue manager. Ensure that you have the authority to connect to the remote queue manager, and ensure that the network is operational. Select Yes if you believe that the problem can be resolved later. Select No if you want to correct the problem now and try again.

AMQ4146
Eclipse cannot create or read the workspace for WebSphere MQ Explorer.

Severity
40: Stop Error

Explanation
To load the WebSphere MQ Explorer, a valid workspace is required.

Response
Ensure that you can write to the Eclipse workspace.

AMQ4147
Eclipse cannot write to the workspace for WebSphere MQ Explorer in <*insert_0*>.

Severity
40: Stop Error

Explanation
To load the WebSphere MQ Explorer, write access to the workspace is required.

Response
Ensure that you can write to the Eclipse workspace.

AMQ4148
The object was created successfully.

Severity
0: Information

Response
Message for information only.

AMQ4149
The request to start the listener was accepted.

Severity
0: Information

Explanation
A user request to start the listener was accepted by WebSphere MQ.

Response
Message for information only.

AMQ4150
The request to stop the listener was accepted.

Severity
0: Information

Explanation
A user request to stop the listener was accepted by WebSphere MQ.

Response
Message for information only.

AMQ4151
The request to start the service was accepted.

Severity
0: Information

Explanation
A user request to start the service was accepted by WebSphere MQ.

Response
Message for information only.

AMQ4152
The request to stop the service was accepted.

Severity
0: Information

Explanation
A user request to stop the service was accepted by WebSphere MQ.

Response
Message for information only.

AMQ4153
WebSphere MQ cannot stop the listener because it is not running.

Severity
10: Warning

AMQ4154
WebSphere MQ cannot start the service because no start command has been specified.

Severity
10: Warning

Response
Ensure that the service has a start command specified.

AMQ4155
WebSphere MQ cannot stop the service because no stop command has been specified.

Severity
10: Warning

Response
Ensure that the service has a stop command specified.

AMQ4156
WebSphere MQ cannot stop the service because the service is not running.

Severity
10: Warning

AMQ4157
WebSphere MQ cannot start the service because the services is already running.

Severity
10: Warning

AMQ4158
WebSphere MQ cannot start the listener because it is already running.

Severity
10: Warning

AMQ4159
WebSphere MQ cannot start the client connection channel because one or more of the properties are incorrectly specified.

Severity
10: Warning

Response
Ensure that the client connection has the correct queue manager name and connection name before trying to start.

AMQ4160
WebSphere MQ cannot process the request because the executable specified cannot be started.

Severity
10: Warning

Explanation
The requested was unsuccessful because the program which was defined to be run to complete the action could not be started.

Reasons why the program could not be started are :-

The program does not exist at the specified location.

The WebSphere MQ user does not have sufficient access to execute the program.

If StdOut or StdErr are defined for the program, the WebSphere MQ user does not have sufficient access to the locations specified.

Response
Check the Queue Manager error logs for further details on the cause of the failure, correct the problem and try again.

AMQ4161
The parameter specified is not valid.

Severity
20: Error

Explanation
The parameter specified when trying to create or alter an object is not valid.

Response
Ensure that valid parameters are specified, then try again.

AMQ4162
The password cannot be cleared.

Severity
0: Information

Response
Try to clear the password again later.

AMQ4163
The password cannot be changed.

Severity
10: Warning

Explanation
The attempt to change the password failed because of an error.

Response
Try a different password

AMQ4164
The password was successfully changed.

Severity
0: Information

Response
Message for information only.

AMQ4165
No password entered in the new password field. No change applied.

Severity
10: Warning

Explanation
You must enter a new password in both the new and confirm password fields.

Response
Enter a new password in the new password field.

AMQ4166
No password entered in the confirm new password field. No change applied.

Severity
10: Warning

Explanation
You must enter a new password in both the new and confirm password fields.

Response
Re-enter the new password in the confirm new password field.

AMQ4167
Passwords do not match. No change applied.

Severity
10: Warning

Explanation
You must enter the same new password in both the new and confirm password fields.

Response
Ensure that the passwords in the new and confirm fields match.

AMQ4168
WebSphere MQ failed to start listening for objects.

Severity
20: Error

Explanation
No objects will be displayed in the currently selected view.

Response
Check the problem determination information, and ensure that WebSphere MQ and the queue manager in question are both running correctly.

AMQ4169
WebSphere MQ failed to set the object filter.

Severity
20: Error

Explanation
The WebSphere MQ Explorer cannot listen for objects, so no objects will be displayed in the currently selected view.

Response
Check the problem determination information, and ensure that WebSphere MQ and the queue manager in question are both running correctly.

AMQ4170
The object name specified is not valid.

Severity
20: Error

Explanation
The object name specified when trying to create or alter an object is not valid.

Response
Ensure that a valid object name is specified, then try again.

AMQ4171
There was an error when communicating with the queue manager.

Severity
20: Error

Explanation
A request for information from the queue manager failed.

Response
Try the operation again. If the error persists, examine the problem determination information to see if any details have been recorded.

AMQ4172
There was an error when trying to set or retrieve information.

Severity
20: Error

Explanation
There was an error when trying to set or retrieve information from the queue manager. This might have happened because you specified incorrect or inconsistent attributes when trying create or update an object.

Response
If this error occurred during object creation or modification, ensure that the attributes specified are correct for this type of object. If the error persists, examine the problem determination information to see if any details have been recorded.

AMQ4173
WebSphere MQ cannot clear one or more Trace and FFST files.

Severity
10: Warning

Explanation
WebSphere MQ cannot clear some files, because of one of the following:

The files are currently in use.

WebSphere MQ Explorer does not have the appropriate access permission.

The trace or errors directories contain user-created subdirectories which WebSphere MQ Explorer cannot delete.

Response
Check that tracing is disabled, and that the WebSphere MQ Explorer has appropriate access permission to delete the Trace and FFST files or remove user created subdirectories.

AMQ4174
FFSTs and Trace were cleared successfully.

Severity
0: Information

Response
Message for information only.

AMQ4175
WebSphere MQ cannot process your request because the value specified is not valid.

Severity
20: Error

Explanation
Only certain combinations and values are valid for the object your are trying to alter or create.

Response
Specify a valid value and try again.

AMQ4176
WebSphere MQ cannot process your request because the object name specified is not valid.

Severity
20: Error

Explanation
Only certain combinations and values are valid for the object your are trying to alter or create. You might also see this message if you have specified a QSG disposition that is not valid or an invalid topic object for a subscription.

Response
Check all values are valid for this type of object and try again. If you have altered the disposition of this object, check that the value is correct. If you are creating a new subscription, check the topic object exists.

AMQ4177
The WebSphere MQ Explorer cannot process your request because the connection to WebSphere MQ is quiescing.

Severity
20: Error

Explanation
The connection to WebSphere MQ is quiescing, so no new information can be queried.

Response
Wait for the connection to end, then try reconnecting.

AMQ4178
WebSphere MQ cannot process your request because there was a disposition conflict detected.

Severity
20: Error

Explanation
A disposition conflict was detected. Ensure that all disposition related fields are correct for this type of object.

Response
Ensure that all disposition related fields are correct for this type of object and try again.

If the error occurred when creating a shared queue check that the Coupling facility structure name on the Storage page has been entered correctly.

If the error occurred while starting a channel that uses a transmission queue with a queue sharing group disposition (QSGDISP) value of SHARED, check that the default channel disposition (DEFCDISP) is set to SHARED or FIXSHARED (and not PRIVATE).

AMQ4179
WebSphere MQ cannot process your request because the string provided was of an incorrect length.

Severity
20: Error

Explanation
A string value has been modified or supplied that is too long or too short when creating or modifying an object.

Response
Check the values being supplied and try again.

Note: If adding exit names on IBM i enter exactly 20 characters, the program name occupies the first 10 characters and the library name occupies the second 10 characters, use blanks to pad to the right if necessary.

AMQ4180
WebSphere MQ cannot process your request because there was a parameter conflict.

Severity
20: Error

Explanation
When creating or modifying an object, the combination of parameters specified is not valid.

Response
Check that the combination specified is valid for the object and try again.

AMQ4181
WebSphere MQ is not responding. Do you want to continue waiting?

Severity
10: Warning

Explanation
WebSphere MQ does not appear to be responding. This could be because of a heavily loaded remote system, or a slow network connection. However there could have been a system failure. Choosing not to continue could leave the WebSphere MQ Explorer in an unknown state, so you should restart it.

Response
If you choose not to continue waiting, restart the WebSphere MQ Explorer, if the problem persists check for problem determination information.

AMQ4182
No objects were found.

Severity
10: Warning

Explanation
The query did not find any objects.

Response
If you were expecting objects to be found, check the problem determination information, and ensure that WebSphere MQ and the queue manager in question are both running correctly.

AMQ4183
Query failed because the queue manager is not in a queue-sharing group.

Severity
10: Warning

Explanation
WebSphere MQ issued a query that required the queue manager to be a member of a queue-sharing group.

Response
Try the operation again, if the problem persists check the problem determination information for more details.

AMQ4184
The channel is not currently active.

Severity
10: Warning

Explanation
The channel was not stopped because it was not currently active.

Response

If attempting to stop a specific instance of a channel, change the connection name or remote queue manager name and try the operation again.

AMQ4185

WebSphere MQ failed to import your settings.

Severity

20: Error

Explanation

One or more of the selected preferences has failed to import your settings.

Response

Try again. If the error persists, examine the problem determination information to see if any details have been recorded.

AMQ4186

WebSphere MQ failed to export your settings.

Severity

20: Error

Response

Try again. If the error persists, examine the problem determination information to see if any details have been recorded.

AMQ4187

WebSphere MQ has successfully imported your settings. (You must restart WebSphere MQ Explorer to apply the imported settings.)

Severity

0: Information

Response

Restart WebSphere MQ explorer to apply the imported settings

AMQ4188

Are you sure that you want to remove queue manager <*insert_0*> from cluster <*insert_1*>?

Severity

10: Warning

Explanation

A confirmation is required before the queue manager is removed from the cluster.

Response

Continue only if you want to permanently remove the queue manager from the cluster.

AMQ4189

The queue manager could not be suspended from the cluster. The operation failed with error <*insert_0*>.

Severity

20: Error

Explanation

The queue manager has not been removed from the cluster.

Response

Try the operation again. If the error persists, examine the problem determination information to see if any information has been recorded.

AMQ4190

An error occurred when clearing the queue manager's REPOS field. The operation failed with error <*insert_0*>.

Severity

20: Error

Explanation

The queue manager has only partially been removed from the cluster. The queue manager has been suspended from the cluster. The REPOS field of the queue manager and the CLUSTER fields of the associated cluster channels have not been cleared.

Response

Try the operation again. If the error persists, examine the problem determination information to see if any information has been recorded.

AMQ4191

An error occurred when clearing the CLUSTER field of channel <*insert_0*>. The operation failed with error <*insert_1*>.

Severity

20: Error

Explanation

The queue manager has only partially been removed from the cluster. The queue manager has been suspended from the cluster and the queue manager's REPOS field has been cleared. Some of the CLUSTER fields of other associated cluster channels might also have been cleared.

Response

To completely remove the queue manager, ensure that all the CLUSTER fields of associated cluster channels are cleared.

AMQ4192

The queue manager could not be removed from a cluster because channel <*insert_0*> is using cluster namelist <*insert_1*>.

Severity

10: Warning

Response

Remove the cluster channel from the cluster namelist. Ensure that you do not inadvertently affect the definitions of other objects using the namelist. Then try removing the queue manager again.

AMQ4193

The information supplied could not be correctly converted to the required code page.

Severity

20: Error

Explanation

All or part of the information entered required conversion to a different code page. One or more characters could not be converted to an equivalent character in the new code page.

Response

Change the characters used, then try the operation again.

AMQ4194

Request failed because the queue manager attempted to use a default transmission queue which is not valid.

Severity

20: Error

Explanation

An MQOPEN or MQPUT1 call specified a remote queue as the destination. The queue manager used the default transmission queue, as there is no queue defined with the same name as the destination queue manager, but the attempt failed because the default transmission queue is not a valid local queue.

Response

Check that the queue manager's default transmission queue property (DefXmitQName) specifies a valid local queue.

AMQ4195

WebSphere MQ Explorer is now in an unknown state and should be restarted. Do you want to restart WebSphere MQ Explorer?

Severity

10: Warning

Explanation

You have chosen not to wait for WebSphere MQ to respond to a request. WebSphere MQ Explorer is therefore in an unknown state and should be restarted.

Response

Restart the WebSphere MQ Explorer and try the operation again. If the problem persists check for problem determination information.

AMQ4196

The command or operation is not valid against the type of object or queue specified

Severity

20: Error

Explanation

You have attempted a command or operation against an object or queue with a type that is not valid for the operation specified. For example, browsing a remote queue; issuing the clear command against a queue with a type that is not QLOCAL; clearing by API calls, a queue who type cannot be opened for input.

Response

Retry the command or operation against an object or queue with a type that is valid for the operation requested.

AMQ4197

An MQOPEN or MQPUT1 call was issued specifying an alias queue as the target, but the BaseObjectName in the alias queue attributes is not recognized as a queue name.

Severity

20: Error

Explanation

An MQOPEN or MQPUT1 call was issued specifying an alias queue as the target, but the BaseObjectName in the alias queue attributes is not recognized as a queue name. This reason code can also occur when BaseObjectName is the name of a cluster queue that cannot be resolved successfully.

Response

Correct the queue definitions.

AMQ4198

Queue manager <*insert_0*> has not been removed from one or more clusters.

If you do not remove the queue manager from the clusters, you might get unexpected errors

Do you want to delete the queue manager without removing it from these clusters?

Severity
10: Warning

Explanation
The user has chosen to delete a queue manager that is currently a member of one or more clusters. The queue manager should first be removed cleanly from these clusters before deleting the queue manager. Other queue managers in the cluster might expect the queue manager to be available.

Response
Remove the queue manager from the clusters it is a member of.

AMQ4199
Queue manager <*insert_0*> is not available for client connection due to an SSL configuration error.

Severity
30: Severe error

Explanation
The user is trying to connect to a remote queue manager using a secure connection.

Response
Check the SSL configuration of the target queue manager and the local SSL trust store.

AMQ4200
There is a problem with the default configuration. Unable to display the Default Configuration window.

Severity
20: Error

Explanation
There is a problem with WebSphere MQ.

Response
Use the 'Details>>' button to show further details about the problem and contact your systems administrator.

AMQ4201
Unable to check if the computer exists.

Severity
20: Error

Explanation
WebSphere MQ was unable to check if the computer name you entered exists on your computer's domain.

Response
Retry the operation, if the problem persists contact your systems administrator.

AMQ4202
Unable to contact the computer <*insert_0*>.

Severity
10: Warning

Explanation
WebSphere MQ was unable to locate a computer with this name on your computer's TCP/IP domain.

Response
Enter a different computer name.

AMQ4203
Unable to set up the default configuration.

Severity
20: Error

Explanation
WebSphere MQ was unable to set up the default configuration. This error may occur if WebSphere MQ is busy with another operation.

Response
Retry the operation. If the problem persists, use the 'Details>>' and 'Print' buttons to record further details about the problem and contact your systems administrator.

AMQ4204
Unable to join the default cluster.

Severity
20: Error

Explanation
WebSphere MQ was unable to join your computer to the default cluster. This error may occur if WebSphere MQ is busy with another operation.

Response
Retry the operation. If the problem persists, use the 'Details>>' and 'Print' buttons to record further details about the problem and contact your systems administrator.

AMQ4205
Unable to allow remote administration of the queue manager.

Severity
20: Error

Explanation
WebSphere MQ was unable change the configuration of your queue manager to allow it to be remotely administered. This error may occur if WebSphere MQ is busy with another operation.

Response
Retry the operation. If the problem persists, use the 'Details>>' and 'Print' buttons to record further details about the problem and contact your systems administrator.

AMQ4206
Unable to prevent remote administration of the queue manager.

Severity
20: Error

Explanation
WebSphere MQ was unable change the configuration of your queue manager to prevent it from being remotely administered. This error may occur if WebSphere MQ is busy with another operation.

Response
Retry the operation. If the problem persists, use the 'Details>>' and 'Print' buttons to record further details about the problem and contact your systems administrator.

AMQ4207
The path specified is not valid.

Severity
20: Error

Response
Check the path specified and try again.

AMQ4208
Show this panel again the next time the queue manager is started?

Severity
0: Information

Explanation
You can choose whether you want the same panel to be shown the next time this queue manager is started, and the default configuration is not complete.

Response
Select whether you want the panel to be shown next time.

AMQ4209
The TCP/IP name of the remote computer must not be your own computer name.

Severity
0: Information

Explanation
You have selected that the repository queue manager is on another computer, but you have entered the name of your own computer.

Response
Enter the correct name of the repository queue manager.

AMQ4210
The command server must be active to complete this operation. Use the WebSphere MQ Services to start it, then retry the operation.

Severity
10: Warning

Explanation
The operation you requested needs the command server to be running.

Response
Use WebSphere MQ Services to start the command server, then retry the operation.

AMQ4211
The computer name entered must be on your local domain (<*insert_0*>).

Severity
10: Warning

Response
Enter the computer name which is on your local domain

AMQ4212
Unable to complete this task because you do not have authority to administer WebSphere MQ.

You must be in the mqm group to administer WebSphere MQ.

Severity
10: Warning

Explanation
Your userid is not authorized to carry out the operation you requested.

Response
Retry the operation on a userid with the required authority, or contact your systems administrator.

AMQ4213
Unable to delete the queue manager <*insert_0*> because it is being used by another program.

Close any program using the queue manager, then click 'Retry'.

Severity
10: Warning

Explanation
WebSphere MQ was unable to delete the old default configuration queue manager because another program is using the queue manager.

Response
Close the programs that are using the queue manager, and click Retry.

AMQ4214
The computer <*insert_0*> is not known on the network.

Severity
10: Warning

Explanation
WebSphere MQ is unable to locate a computer with this name on your network.

Response
Enter a different computer name.

AMQ4215
Upgrade of the default configuration was canceled.

Severity
10: Warning

Explanation
You pressed 'Cancel' while running the default configuration wizard to upgrade the default configuration.

Response
None

AMQ4216
The WebSphere MQ services component does not have the authority it requires.

Severity
10: Warning

AMQ4217
The MQSeriesServices component does not have the authority to create the default configuration.

Severity
10: Warning

AMQ4250
No nickname supplied - supply one.

Severity
10: Warning

Explanation
Requires to enter the user nick name in the text box

Response
Enter the nickname in the text box

AMQ4251
Cannot Initialise WinSock - TCP/IP may not be installed. Install TCP/IP and try again

Severity
20: Error

Explanation
Postcard was not able to initialize the interface to TCP/IP.

Response
Check that TCP/IP has been installed successfully. If the problem persists, refer to your systems administrator.

AMQ4252
Cannot Find WinSock - TCP/IP may not be installed. Install TCP/IP and try again.

Severity
20: Error

Explanation
Postcard was not able to find the interface to TCP/IP.

Response
Check that TCP/IP has been installed successfully. If the problem persists, refer to your systems administrator.

AMQ4253
Cannot get fully qualified TCP/IP domain name - Ensure that the TCP/IP protocol is configured.

Severity
20: Error

Explanation
Postcard was not able to determine the TCP/IP domain name for your computer.

Response
Check that TCP/IP has been installed successfully. If the problem persists, refer to your systems administrator.

AMQ4254
Failed to Allocate System Memory - Contact your system administrator.

Severity
20: Error

Explanation
Postcard was not able to allocate enough memory to run correctly.

Response
Close other programs to release system memory. If the problem persists, refer to your systems administrator.

AMQ4255
Supply a user name with which you wish to communicate.

Severity
10: Warning

Explanation
Requires to enter a user nick name in the To text box.

Response
Enter the user nickname in the To text box

AMQ4256
Supply <*insert_0*>s computer name (this must be a TCP/IP name).

Severity
10: Warning

Explanation
Requires to enter the mail box computer name on the On field

Response
Enter the mail box computer name or queue manager name on the On text box

AMQ4257
The call MQCONN failed while preparing for a Put operation,

with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to connect to the queue manager in order to send the postcard. This error may occur if WebSphere MQ is busy with another operation.

Response
Try to send the postcard again. If the problem persists contact your systems administrator.

AMQ4258
The call MQOPEN failed while preparing for a Put operation,

with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to open a queue in order to send the postcard. This error may occur if WebSphere MQ is busy with another operation.

Response
Try to send the postcard again. If the problem persists contact your systems administrator.

AMQ4259
The call MQCLOSE failed while preparing for a Put operation,

with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to close the queue after sending the postcard. This error may occur if WebSphere MQ is busy with another operation.

Response
If the problem persists contact your systems administrator.

AMQ4260
The call MQDISC failed while preparing for a Put operation,

with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to disconnect from the queue manager after sending the postcard. This error may occur if WebSphere MQ is busy with another operation.

Response
If the problem persists contact your systems administrator.

AMQ4261
The call MQPUT failed with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to send the postcard by putting its data to the queue. This error may occur if WebSphere MQ is busy with another operation.

Response
Try to send the postcard again. If the problem persists contact your systems administrator.

AMQ4262
The call MQCONN failed while preparing for a Get operation,

with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to connect to the queue manager in order to receive postcards. This error may occur if WebSphere MQ is busy with another operation.

Response
Restart Postcard. If the problem persists contact your systems administrator.

AMQ4263
The call MQOPEN failed while preparing for a Get operation,

with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to open a queue in order to send the postcard. This error may occur if WebSphere MQ is busy with another operation.

Response
Restart Postcard. If the problem persists contact your systems administrator.

AMQ4264
The call MQCLOSE failed while preparing for a Get operation,

with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to close the queue after receiving postcards. This error may occur if WebSphere MQ is busy with another operation.

Response
If the problem persists contact your systems administrator.

AMQ4265
The call MQDISC failed while preparing for a Get operation,

with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to disconnect from the queue manager after receiving postcards. This error may occur if WebSphere MQ is busy with another operation.

Response
If the problem persists contact your systems administrator.

AMQ4266
Enter the message that you want to send to <*insert_0*>.

Severity
10: Warning

Response
Enter the message in the Message text field.

AMQ4267
The call MQGET failed with Completion Code [<*insert_0*> (<*insert_1*>)], Reason Code [<*insert_2*> (<*insert_3*>)].

Severity
20: Error

Explanation
An error occurred when Postcard tried to receive postcards by getting its data from the queue. This error may occur if WebSphere MQ is busy with another operation.

Response
Restart Postcard. If the problem persists contact your systems administrator.

AMQ4268
Postcard is unable to contact the queue manager on the remote computer.

Verify that the default configuration is running on the remote computer.

Severity
20: Error

Explanation
The mail box queue manager in the On text box not reachable.

Response
Verify that the default configuration is running on the remote computer.

AMQ4269
Unable to run Postcard because you do not have authority to use WebSphere MQ.

You must be in the mqm group to use WebSphere MQ.

Severity
20: Error

Explanation
The mail box queue manager in the On text box not reachable.

Response
Use Postcard on a user Id with the required authority, or contact your systems administrator.

AMQ4270
Postcard is unable to send messages to the remote computer. Postcard can only exchange messages with computers that are on the same TCP/IP domain as this computer.

Severity
20: Error

Explanation
Unable to send messages to the remote computer

Response
Use default configuration application to add the remote computer to the same cluster.

AMQ4271
Unable to open a local queue called <*insert_0*> on the mailbox queue manager <*insert_1*>.

Use WebSphere MQ Explorer to create the queue, then restart Postcard.

Severity
20: Error

Explanation
Postcard was unable to automatically create the queue it uses on the queue manager.

Response
Use WebSphere MQ Explorer to create the queue, and restart Postcard.

AMQ4272
The mailbox queue manager <*insert_0*> does not exist on this computer.

Severity
20: Error

Explanation
The mailbox queue manager name specified after the '-m' parameter to Postcard does not exist on this computer.

Response
Restart Postcard specifying the name of a queue manager that does exist on this computer.

AMQ4273
Unable to contact the target mailbox <*insert_0*>.

Severity
10: Warning

Explanation
Postcard was unable send the message as it could not contact the target mailbox.

Response
Click 'Retry' to attempt to send the message again, otherwise click 'Cancel'.

AMQ4274
Postcard has detected that <*insert_0*> is the name of a computer and a queue manager.

Severity
10: Warning

Explanation
Postcard has detected that the destination mailbox name is the name of a computer and of a queue manager.

Response
Select whether you want to send the message to the computer or the queue manager with this name, then click OK.

AMQ4300
Supply some text in order for the MQPUT(1) operation to succeed.

Explanation
No text has been supplied for the user so that the MQPUT or MQPUT1 operation can proceed.

Response
Supply some text in the editable area so that the MQPUT or MQPUT1 operation can proceed.

AMQ4301
Supply some text in order for the MQPUT operation to succeed.

Explanation
No text has been supplied for the user so that the MQPUT operation may proceed.

Response
Supply some text in the editable area so that the MQPUT may proceed.

AMQ4302
Supply some text in order for the MQPUT1 operation to succeed.

Explanation
No text has been supplied for the user so that the MQPUT1 operation may proceed.

Response
Supply some text in the editable area so that the MQPUT1 may proceed.

AMQ4303
The command server for the queue manager [%s] is not started. Start the command server and try again.

Explanation
In order for the API Exerciser to function, a command server must be running.

Response
Either start the command server from the MQServices application or run strmqcsv <Queue Manager> from the command line.

AMQ4304
API Exerciser cannot enumerate objects for queue manager [%s].

Explanation
The API Exerciser encountered a problem trying to enumerate queues.

Response
Ensure that the command server is running (from the Service application) and that there are queues configured for the queue manager.

AMQ4305
There are no queue managers present in the system. Create one and try again.

Explanation
The API Exerciser could not find any queue managers on the system.

Response
Use the Services application to create one or run crtmqm <Queue Manager>.

AMQ4306
Memory allocation failure. Stop some other applications and try again.

Explanation
There are not sufficient system resources available in the system to satisfy the running of API Exerciser.

Response
Shut some other applications down and try running the API Exerciser again.

AMQ4307
API Exerciser encountered a COM failure and cannot continue. Ensure that WebSphere MQ has been correctly installed and configured and that your user id. is a member of the mqm group.

Explanation
When the API Exerciser started, it was unable to make a COM connection to WebSphere MQ Services.

Response
Ensure that WebSphere MQ has been correctly installed and configured, and that your user ID is a member of the mqm group. If the problem persists, refer to your systems administrator.

AMQ4308
API Exerciser cannot continue. Ensure that the userid you are using is a member of the mqm group.

Explanation
None.

Response
None.

AMQ4309
API Exerciser cannot continue. Ensure that the userid you are using is a member of the Administrator group.

Explanation
None.

Response
None.

AMQ4350
Setup cannot continue; a later version of this product is installed.

Explanation
Installation detected that a version of this product later than version 5.3 is already installed on the computer.

Response
Do not attempt to install version 5.3 when a later version is already installed.

AMQ4351
Uninstallation cannot continue; uninstallation is already running.

Explanation
An attempt was made to run two copies of uninstallation at once.

Response
Run only one copy of uninstallation at a time.

AMQ4352
Setup cannot continue; a supported version of Windows is required.

Explanation
None.

Response
None.

AMQ4353
Setup cannot continue; '%s' is not an Administrator.

Explanation
The user running installation does not have administrator authority.

Response
Log off and log back on using a user ID with administrator authority.

AMQ4354
No repository computer name entered.

Explanation
None.

Response
None.

AMQ4355
Repository computer name is not valid.

Explanation
None.

Response
None.

AMQ4356
Enter a remote computer name.

Explanation
None.

Response
None.

AMQ4357
Registration failed for file '%s' (code 0x%8.8lx).

Explanation
None.

Response
None.

AMQ4358
Unregistration failed for file '%s' (code 0x%8.8lx).

Explanation
None.

Response
None.

AMQ4359
Unable to register file '%s'.

Explanation
None.

Response
None.

AMQ4360
Unable to unregister file '%s'.

Explanation
None.

Response
None.

AMQ4361
Uninstall cannot continue; Administrator logon required.

Explanation
None.

Response
None.

AMQ4362
Failed to create the default configuration.

Explanation
None.

Response
None.

AMQ4363
Setup could not detect the Windows NT Service Pack level (Service Pack 3 or later is required). Is Service Pack 3 or later installed?

Explanation
None.

Response
None.

AMQ4364
Setup could not detect the Windows NT Service Pack level (Service Pack 6a or later is required). Is Service Pack 6a or later installed?

Explanation
None.

Response
None.

AMQ4365
Setup cannot continue because Service Pack 3 is not installed.

Explanation
None.

Response
None.

AMQ4366
Setup cannot continue because Service Pack 6a or later is not installed.

Explanation
None.

Response
None.

AMQ4367
Setup cannot continue because Internet Explorer Version 4.01 SP1 is not installed.

Explanation
None.

Response
None.

AMQ4368
Select at least one component to proceed.

Explanation
None.

Response
None.

AMQ4369
The 'Web Administration Server' component requires the 'Server' component.

Explanation

Response

AMQ4370
Uninstallation of the 'Server' component requires uninstallation of the 'Web Administration Server' component.

Explanation
None.

Response
None.

AMQ4371
The 'Documentation in Other Languages' component requires the 'Documentation in English' component.

Explanation
None.

Response
None.

AMQ4372
Uninstallation of the 'Documentation in English' component requires uninstallation of the 'Documentation in Other Languages' component.

Explanation
None.

Response
None.

AMQ4373
There is not enough space on drive %s (program files) to install these components. Free up some disk space or modify your selections

Explanation
None.

Response
None.

AMQ4374
There is not enough space on drive %s (data files) to install these components. Free up some disk space or modify your selections

Explanation
None.

Response
None.

AMQ4375
The program files top-level folder is not valid.

Explanation
The program files top-level folder is not a valid path.

Response
Enter a valid path.

AMQ4376
The data files top-level folder is not valid.

Explanation
The data files top-level folder is not a valid path.

Response
Enter a valid path.

AMQ4377
The log files folder is not valid.

Explanation
The log files folder name is not a valid path.

Response
Enter a valid path.

AMQ4378
A root folder is not allowed for the program files top-level folder.

Explanation
WebSphere MQ cannot be installed in a root folder, for example 'C:\'.

Response
Enter a non-root folder.

AMQ4379
A root folder is not allowed for the data files top-level folder.

Explanation
WebSphere MQ cannot be installed in a root folder, for example 'C:\'.

Response
Enter a non-root folder.

AMQ4380
A root folder is not allowed for the log files folder.

Explanation
WebSphere MQ cannot be installed in a root folder, for example 'C:\'.

Response
Enter a non-root folder.

AMQ4381
here is not enough space on drive %s (log files) to install these components. Free up some disk space or modify your selections

Explanation
None.

Response
None.

AMQ4382
Unable to create or replace folder '%s'

Explanation
None.

Response
None.

AMQ4385
Unknown language specified ('%s')

Explanation
None.

Response
None.

AMQ4386
Codepage (%d) for specified language not available.

Explanation
None.

Response
None.

AMQ4387
Before Setup can display help, this computer's help system needs upgrading to HTML Help 1.3. Would you like to upgrade now? (You might need to restart the computer.)

Explanation
None.

Response
None.

AMQ4388
WebSphere MQ Setup or uninstallation is already running.

Explanation
None.

Response
None.

AMQ4389
Setup could not create a local 'mqm' group (code %d).

Explanation
An error occurred creating a local user group called 'mqm'.

Response
Review the installation log file for details of any problems. If the error persists, contact your systems administrator.

AMQ4390
Setup could not create a global 'Domain mqm' group (code %d).

Explanation
An error occurred creating a local user group called 'mqm'.

Response
Review the installation log file for details of any problems. If the error persists, contact your systems administrator.

AMQ4391
Setup could not find the global 'Domain mqm' group.

Explanation
The global 'mqm ' group was created, but could not then be found.

Response
Review the installation log file for details of any problems. If the error persists, contact your systems administrator.

AMQ4392
Setup could not add the global 'Domain mqm' group to the local 'mqm' group (code %d).

Explanation
An error occurred adding the global 'mqm' group to the local 'mqm' group.

Response
Review the installation log file for details of any problems. If the error persists, contact your systems administrator.

AMQ4393
No ports were specified; no listeners will be created.

Explanation
None.

Response
None

AMQ4394
No queue managers are selected for remote administration.

Explanation
None.

Response
None.

AMQ4395
One or more 'Server' component prerequisites were not selected; the component cannot be installed.

Explanation
None.

Response
None.

AMQ4396
One or more prerequisite upgrades were not selected; WebSphere MQ will not operate correctly.

Explanation
None.

Response
None.

AMQ4397
Cannot install on a network drive (drive %s).

Explanation
None.

Response
None.

AMQ4400
Explorer cannot administer the queue manager because the queue <*insert_0*> is not defined.

Severity
10: Warning

Explanation
Explorer uses the queue <*insert_0*> to administer queue managers.

Response
Define the queue <*insert_0*> and retry.

AMQ4401

Explorer cannot administer the queue manager because the user is not authorised to open the queue <*insert_0*>.

Severity

10: Warning

Explanation

Explorer uses the queue <*insert_0*> to administer this queue manager.

Response

Allow Explorer to open the queue <*insert_0*> and retry.

AMQ4402

The queue <*insert_0*> could not be opened for reason <*insert_1*>.

Severity

10: Warning

Explanation

Explorer uses the queue <*insert_0*> to administer this queue manager.

Response

Allow Explorer to open the queue <*insert_0*> and retry.

AMQ4403

The queue manager you are connecting to is at a higher command level than the intermediate queue manager you are using, which will cause some operations not to work. Are you sure that you want to show the destination queue manager in the folder anyway?

Severity

10: Warning

Explanation

You are making a connection to a remote queue manager which is at a command level higher than the intermediate queue manager you are trying to use. This means that errors will occur when selecting new items such as Application Connections or queue status.

Response

Select Yes if you want to continue to use the remote queue manager with this intermediate queue manager, even though the command levels are inconsistent. Select No to chose a different intermediate queue manager.

AMQ4404

The queue manager <*insert_0*> is the only full repository in cluster <*insert_1*> and there are still partial repository queue managers defined. Removing this queue manager from the cluster prevents further repository actions from being run. Are you sure that you want to remove this queue manager?

Severity

10: Warning

Explanation

To be able to display cluster information, the clustering component of the WebSphere MQ Explorer requires at least one full repository to be selected as the source. Removing the last full repository will prevent the display of cluster members, and hence will prevent cluster actions being run on these full repositories.

Response

Select Yes if you want to remove the full repository even though it will prevent access to remaining partial repository information.

AMQ4405

An unexpected error occurred connecting to the JNDI service provider.

The following message contains text from the JNDI service provider which might not be translated.

Error <*insert_0*> performing JNDI operation <*insert_1*> on object name <*insert_2*>.

Severity
30: Severe error

Explanation
An unexpected JNDI error prevented the operation from completing.

Response
Check for FFSTs to determine the reason for the error. If symptoms persist, contact your Systems Administrator.

AMQ4406
The connection could not be made to the JNDI service provider because the specified security credentials (distinguished name and password) are not valid for this service provider.

Severity
20: Error

Explanation
Either the distinguished name or password is not valid for the service provider

Response
Correct the security credentials and try again.

AMQ4407
The Provider URL was not supplied.

Severity
20: Error

Explanation
The Provider URL must be supplied when opening an Initial Context.

Response
Supply the Provider URL.

AMQ4408
The NAME was missing from the JMS Administration data file.

Severity
20: Error

Response
Check for FFSTs to determine the reason for the error.

AMQ4409
A context with the nickname <*insert_0*> already exists.

Severity
20: Error

Explanation
Nicknames for each context in the tree must be unique.

Response
Choose a different nickname for this context.

AMQ4410
Object type <*insert_0*> is not recognised when retrieving details for attribute <*insert_1*>.

Severity
20: Error

Explanation
The object ID is not valid.

Response
Ensure that only supported object types are used.

AMQ4411
Object type <*insert_0*> is not recognised when loading objects from context <*insert_1*>.

Severity
20: Error

Explanation
The object class is not valid.

Response
Ensure that only supported object types are used.

AMQ4412
Unexpected Exception: <*insert_0*> message <*insert_1*>.

Severity
20: Error

Explanation
An unexpected error occurred.

Response
Check for FFSTs to determine the reason for the error.

AMQ4413
The context <*insert_0*> could not be removed, because it was not empty.

Severity
20: Error

Explanation
A context can only be removed if it is empty.

Response
Remove the contents of the context and try again.

AMQ4414
An unexpected error occurred when connecting to the JNDI service provider.

The following message contains text from the JNDI service provider which might not be translated.

Error <*insert_0*> because of <*insert_3*> performing JNDI operation <*insert_1*> on object name <*insert_2*>.

Severity
30: Severe error

Explanation
An unexpected JNDI error prevented the operation from completing.

Response
Check for FFSTs to determine the reason for the error. If symptoms persist contact your Systems Administrator.

AMQ4415
The object could not be created because an object with the name <*insert_0*> already exists.

Severity
20: Error

Explanation

An object with the same name already exists in JNDI. Note that the existing object might be of a different type to the one being created as Connection Factories, Destinations and other JNDI objects all share the same namespace within a particular JNDI context. To locate the existing object, select the JMS context tree node to display all objects within that JNDI location.

Response

Choose a different name for the new object, or delete the existing object.

AMQ4416

The object <*insert_0*> could not be created because you do not have authority to create objects, or there is no connection to the context.

Severity

20: Error

Explanation

If the JNDI service provider is LDAP then the connection might not have a sufficient level of security to create objects.

If the JNDI service provider is a file system then the bindings file might be read-only,

or there is no connection to the context.

Response

Connect to the JNDI service provider with the correct level of security, or ensure the permissions on the bindings file are correct and try again.

AMQ4417

The Local address could not be set to the value <*insert_0*>.

Severity

20: Error

Explanation

The Local address must be a valid address in the form ip_address(port-number), where the port number can be a specific port, a range of ports (low-port,high-port), or can be omitted. A host name can be specified instead of an IP address.

Response

Correct the Local address and try again.

AMQ4418

The SSL Peer name could not be set to the value <*insert_0*>.

Severity

20: Error

Explanation

SSL Peer name must be a valid Distinguished Name.

Response

Enter a valid SSL Peer name.

AMQ4419

The JNDI context was opened out of order.

Severity

20: Error

Explanation

A context which is already open cannot be opened again.

Response

Check for FFSTs to determine the reason for the error.

AMQ4420
The JNDI context was closed out of order.

Severity
20: Error

Explanation
A context which is already closed cannot be closed again.

Response
Check for FFSTs to determine the reason for the error.

AMQ4421
The connection could not be made to the JNDI service provider. This could be either because the physical connection has been broken, or the distinguished name in the provider URL or the distinguished name provided for the security credentials is not valid.

Severity
20: Error

Explanation
The name provided must be a properly formed distinguished name, valid on the specified JNDI service provider.

Response
Correct the distinguished name and try again.

AMQ4422
There is a communication error connecting to the JNDI service provider with the provider URL <*insert_0*>.

Severity
20: Error

Explanation
The connection to the JNDI service provider has timed out.

Response
Check the connection information and ensure that the service provider is running at the remote end and try again.

AMQ4423
The object <*insert_0*> could not be deleted because you do not have authority to delete objects.

Severity
20: Error

Explanation
If the JNDI service provider is LDAP then the connection might not have a sufficient level of security to delete objects.

If the JNDI Service provider is a file system then the bindings file might be read-only.

Response
Connected to the JNDI service provider with the correct level of security or ensure the permissions on the bindings file are correct and try again.

AMQ4424
The requested level of security is not supported by the JNDI service provider.

Severity
20: Error

Explanation
The level of security requested (none, simple or CRAM_MD5) is not supported by the JNDI service provider being used.

Response
Either change the level of security requested or the JNDI service provider and try again.

AMQ4425
It is not clear to which queue manager the value of the <*insert_0*> field on the <*insert_1*> page refers.

* Ensure that the queue manager is in WebSphere MQ Explorer.

* Ensure that the queue manager is running.

* Ensure that WebSphere MQ Explorer is connected to the queue manager.

* Ensure you have authority to list queues on the queue manager

* If there are two queue managers with the same name in WebSphere MQ Explorer, use the <*insert_0*> Select button to specify the queue manager again.

Severity
20: Error

Explanation
WebSphere MQ Explorer needs to know exactly which queue manager to query to populate the object selection dialog.

Response
If the queue manager name is ambiguous, use the selection button to choose a running queue manager, before selecting the object.

AMQ4426
The location <*insert_0*> cannot be resolved.

Severity
20: Error

Explanation
The specified location could not be found because it is not bound.

Response
Ensure that the details for the JNDI context are correct and the context itself is accessible. Try again.

AMQ4427
The JNDI service provider cannot be found

Severity
20: Error

Explanation
A JNDI service provider has been entered that is not valid, or it cannot be found in the CLASSPATH.

Response
Correctly specify the JNDI service provider and try again.

AMQ4428
There is an error connecting to the JNDI service provider with the provider URL <*insert_0*>.

The host name or IP address is not correct.

Severity
20: Error

Explanation
The connection to the JNDI service provider has timed out due to an incorrect host name or IP address.

Response
Correct the host name or IP address and try again.

AMQ4429
There is an error connecting to the JNDI service provider with the provider URL <*insert_0*>.

The host name or port number is not correct or the remote server is not running.

Severity
20: Error

Explanation
The connection to the JNDI service provider has timed out due an incorrect host name or port number, or the remote server is not running.

Response
Check the host name and port number and ensure that the remote service provider is running.

AMQ4430
There is an error connecting to the JNDI service provider with the provider URL <*insert_0*>.

The Local area network (LAN) is not available.

Severity
20: Error

Explanation
The connection to the JNDI service provider has timed out due to the LAN not being available.

Response
Ensure the LAN is available and try again.

AMQ4431
The object <*insert_0*> could not be updated as you do not have authority to update objects.

Severity
20: Error

Explanation
If the JNDI service provider is LDAP, then the connection might not have a sufficient level of security to update objects.

If the JNDI Service provider is a file system, then the bindings file might be read-only.

Response
Connected to the JNDI service provider with the correct level of security, or ensure the permissions on the bindings file are correct and try again.

AMQ4432
There is a communication error with the JNDI service provider.

Severity
20: Error

Explanation
The connection to the JNDI service provider has timed out.

Response
Ensure that the LAN is available and that the remote service provider is running, then try again.

AMQ4433
The object <*insert_0*> could not be renamed because you do not have authority to rename objects.

Severity
20: Error

Explanation
If the JNDI service provider is LDAP, then the connection might not have a sufficient level of security to rename objects.

If the JNDI Service provider is a file system then the bindings file might be read-only.

Response
Connect to the JNDI service provider with the correct level of security, or ensure the permissions on the bindings file are correct and try again.

AMQ4434
The object <*insert_0*> could not be renamed to <*insert_1*> because the name already exists.

Severity
20: Error

Explanation
Names within the JNDI namespace must be unique.

Response
Choose another name and try again.

AMQ4435
The field <*insert_0*> must start with the prefix <*insert_1*>

Severity
20: Error

Explanation
The name entered must start with the particular prefix.

Response
Correct the name and try again.

AMQ4436
The <*insert_0*> on the <*insert_1*> page cannot be <*insert_2*> when the <*insert_3*> on the <*insert_4*> page is <*insert_5*>.

Severity
20: Error

Explanation
The attributes are inconsistent.

Response
Change one or both of the attributes to make them consistent.

AMQ4437
Unknown event; type <*insert_0*>.

Severity
20: Error

Explanation
The JMS Administration plug-in encountered an unexpected event.

Response
Check for FFSTs to determine the reason for the error.

AMQ4438
The value <*insert_3*> from the parameter <*insert_0*> <*insert_1*> of class <*insert_2*> cannot be converted into a URL.

Severity
20: Error

Explanation
The JMS Administration plug-in encountered an unexpected URL string.

Response
Check for FFSTs to determine the reason for the error.

AMQ4439
The last non-blank character of <*insert_0*> must be an asterisk.

Severity
20: Error

Explanation
The name entered must end with an asterisk.

Response
Correct the name and try again.

AMQ4440
The following error was encountered when setting the field <*insert_0*>.

<*insert_1*>

Severity
20: Error

Explanation
A JMS exception was generated when setting the SSL CRL

Response
Check that all the URLs in the SSL CRL field are in the format "ldap://host".

AMQ4441
The type of the object underlying the JMS Parameter <*insert_0*> <*insert_1*> is unexpected: <*insert_2*>.

Severity
20: Error

Explanation
The JMS Administration plug-in encountered an unexpected object type.

Response
Check for FFSTs to determine the reason for the error.

AMQ4442
Unexpected JMS Exception: pcfid: <*insert_0*> <*insert_1*>, object type: <*insert_2*>, JMS error <*insert_3*> <*insert_4*>.

Severity
20: Error

Explanation
The JMS Administration plug-in encountered an unexpected JMS error.

Response
Check for FFSTs to determine the reason for the error.

AMQ4443
One or more JNDI errors prevented objects being retrieved from the namespace. The last of these errors was <*insert_0*> for the object <*insert_1*>.

Severity
30: Severe error

Explanation
An unexpected JNDI error prevented the operation from completing. The objects might have been damaged and cannot be retrieved from the namespace. Damaged objects are shown in WebSphere MQ Explorer

Response
Either delete the object (using the Explorer), or repair it using some other tool.

AMQ4444
One or more JNDI errors prevented objects being looked up from the namespace. The last of these errors was <*insert_0*> for the object <*insert_1*>.

The JNDI service provider returned the following message text:

<*insert_2*>.

Severity
30: Severe error

Explanation
An unexpected JNDI error prevented the operation from completing. The objects might have been damaged and cannot be retrieved from the namespace. Damaged objects are shown in WebSphere MQ Explorer

Response
Either delete the object (using the Explorer), or repair it using some other tool.

AMQ4445
The following error, reported by JNDI, prevented the transport being changed for the object: <*insert_1*>.

<*insert_0*>.

Severity
30: Severe error

Explanation
The objects might have properties which prevent the transport being changed.

Response
Before trying to change the transport, change any conflicting properties.

AMQ4446
You are about to remove the Initial context <*insert_0*> (<*insert_1*>) from WebSphere MQ Explorer. Are you sure that you want to continue?

Severity
0: Information

Explanation
If you remove this Initial context, it will no longer be displayed in WebSphere MQ Explorer. The context itself and its contents, will not be deleted.

Response
Continue only if you want to remove the context from WebSphere MQ Explorer.

AMQ4447
Are you sure that you want to delete the JMS object <*insert_0*> (<*insert_1*>)?

Severity
0: Information

Explanation
The JMS object will be permanently removed from the JMS Context.

Response
Continue only if you want to permanently delete the object.

AMQ4448
The <*insert_0*> on the <*insert_1*> page cannot be specified when the <*insert_2*> on the <*insert_3*> page has not been specified.

Severity
20: Error

Explanation
The attributes are inconsistent.

Response
Change one or both of the attributes to make them consistent.

AMQ4449
The factory class location <*insert_0*> is not valid.

Severity
20: Error

Explanation
The factory class location must be in a URL format.

Response
Remove the Initial context from WebSphere MQ Explorer and add it again.

AMQ4450
This operation is not supported. The following message contains text from the JNDI service provider which might not be translated:

<*insert_0*>

Use this message to help you diagnose the problem.

Severity
20: Error

Explanation
The JNDI provider does not support the operation performed. One common problem is trying to connect without a password.

Response
Determine and solve the problem from the JNDI error message and try again.

AMQ4451
The <*insert_0*> property on the JMS object <*insert_1*> is set to <*insert_2*> but WebSphere MQ Explorer is not connected to a queue manager with that name.

Severity
20: Error

Explanation
To create the appropriate object on the queue manager, WebSphere MQ Explorer must be connected to it.

Response
Add the required queue manager to WebSphere MQ Explorer and ensure that it is connected before attempting this operation again.

AMQ4452

The coupling-facility structure name specified in the queue definition for this queue is not defined in the CFRM data set, or is not the name of a list structure.

Severity

20: Error

Explanation

An MQOPEN or MQPUT1 call was issued to access a shared queue, but the call failed because the coupling-facility structure name specified in the queue definition is not defined in the CFRM data set, or is not the name of a list structure.

Response

Modify the queue definition to specify the name of a coupling-facility list structure that is defined in the CFRM data set.

AMQ4453

The storage class defined for this queue does not exist.

Severity

20: Error

Explanation

The MQPUT or MQPUT1 call was issued, but the storage-class object defined for the queue does not exist.

Response

Create the storage class object required by the queue, or modify the queue definition to use an existing storage class. The name of the storage class object used by the queue is specified by the StorageClass queue attribute.

AMQ4454

There is an error associated with this channel.

Severity

20: Error

Explanation

A possible error cause is that the channel references a host name that cannot be resolved.

Response

Ensure that all of the properties for the channel have been defined correctly. Ensure that the channel references a host name that can be resolved.

AMQ4455

The Distinguished Name specified is not valid.

Severity

20: Error

Response

Ensure that a valid Distinguished Name is specified.

AMQ4456

The Db2 subsystem is currently not available.

Severity

20: Error

Explanation

An MQOPEN, MQPUT1, or MQSET call was issued to access a shared queue, but the call failed because the queue manager is not connected to a Db2 subsystem. As a result, the queue manager is unable to access the object definition relating to the shared queue. A possible cause for this error is that the Db2 subsystem is being restarted.

Response

Configure the Db2 subsystem so that the queue manager can connect to it. Ensure that the Db2 subsystem is available and running.

AMQ4457

The value <*insert_0*> from attribute <*insert_1*> on JMS object <*insert_2*> is not a valid name for an MQ object.

Severity

20: Error

Explanation

The value of the specified attribute either contains invalid characters or is an invalid length for an MQ object name.

Response

Modify the attribute value by removing any invalid characters or reducing the length.

AMQ4458

The property <*insert_0*> on JMS object <*insert_1*> could not be retrieved or updated.

Severity

20: Error

Explanation

An error occurred while requesting or updating the value of a property on a JMS object.

Response

Check for FFST information to determine the reason for the error. If symptoms persist, contact your Systems Administrator.

AMQ4459

The <*insert_0*> property on the JMS object <*insert_1*> is set to <*insert_2*> but no known queue managers of that name support the creation of administrative topic objects.

Severity

20: Error

Explanation

To create the appropriate object on the queue manager, it must support the creation of administrative topic objects.

Response

Either add a queue manager of the appropriate name and that supports the creation of administrative topics to WebSphere MQ Explorer, or modify the JMS object property. Try the operation again.

AMQ4460

The default remote administration listener LISTENER.TCP was created successfully.

Severity

0: Information

Response

Message for information only.

AMQ4461

The default remote administration listener LISTENER.TCP could not be created.

Severity

10: Warning

Explanation

A problem occurred when issuing a command to the command server to create the listener.

Response
Check that the command server is running on the queue manager and try again. If symptoms persist contact your System Administrator.

AMQ4462
Successfully added queue manager <*insert_0*>.

Severity
0: Information

Explanation
The requested queue manager was successfully added to the list of known queue managers in the WebSphere MQ Explorer.

Response
Message for information only.

AMQ4463
The <*insert_0*> attribute on JMS object <*insert_1*> is set to <*insert_2*> but this is not a valid name for an MQ Queue Manager.

Severity
20: Error

Explanation
The attribute must only contain valid characters and be of the appropriate length for an MQ Queue Manager name.

Response
Modify the attribute to the name of a real MQ Queue Manager.

AMQ4464
An error occurred while trying to connect to the queue manager. WebSphere MQ Explorer could not determine the name of the queue manager so it cannot be added.

Severity
20: Error

Explanation
Queue manager names must be determined before adding them to the WebSphere MQ Explorer. Where an asterisk (*) is used to connect, the Queue Manager must be available so that the queue manager name can be determined.

Response
Ensure the required queue manager is available before attempting this operation again, or make the queue manager name explicit rather than using an asterisk (*).

AMQ4465
New attributes have been added to WebSphere MQ Explorer objects. Your existing user-defined schemes have not been updated. If you want your user-defined schemes to contain these new attributes, you must manually add the new attributes.

Severity
0: Information

Response
Message for information only.

AMQ4466
Successfully connected to the queue manager <*insert_0*>. As the required queue manager name <*insert_1*> starts with an asterisk (*), there might be multiple queue managers that could result from the same connection. Are you sure that you want to add this queue manager?

Severity
0: Information

Explanation
The queue manager name used to connect starts with an asterisk (*). This means that the same connection details could be used to connect to multiple queue managers.

Response
Add the queue manager specified if it is the one you required.

AMQ4467
The filter has not been removed because it is still required by other plug-ins.

Severity
10: Warning

Explanation
Other plug-ins have responded to the attempted removal of this filter by indicating that they are still using it.

Response
Ensure that the other plug-ins have finished using the filter before trying to delete it again.

AMQ4468
The filter named <*insert_0*> is used by the following automatic sets:<*insert_1*> Are you sure that you want to delete this filter?

Severity
10: Warning

Explanation
A confirmation is required before the specified filter is deleted. The name is provided in the message.

Response
Continue only if you want to permanently delete the filter.

AMQ4469
The automatic set <*insert_0*> no longer has any filters to decide its membership.

Severity
10: Warning

Explanation
The only filter that this set was using has been deleted. An automatic set needs at least one filter to determine which objects should be members of the set.

Response
Press OK to edit this set and in the Edit Set dialog, select one or more filters to use with this set.

AMQ4470
The Provider Version is not in the correct form.

Severity
20: Error

Explanation
The Provider Version consists of up to 4 groups of digits separated with periods (.)but not ending with one, 63, 1.2 or 1.2.34.56 for example. Alternatively you can enter the word 'unspecified'.

Response
Correct the provider version and try again.

AMQ4471
Are you sure that you want to delete the set named <*insert_0*>?

Note that deleting a set does not delete its members.

Severity
10: Warning

Explanation
A confirmation is required before the specified set is deleted.

Response
Continue only if you want to permanently delete the set.

AMQ4472
The WMQ_Schemes.xml file used to save schemes is incomplete.

A backup copy of this file has been made:

<insert_0>.

Where possible, user-defined schemes from this file have been extracted and retained,

but it is possible that some have been lost.

Severity
10: Warning

Explanation
When reading in schemes from the WMQ_Schemes.xml file, some required information was missing.

Response
Re-create user-defined schemes where necessary. Refer to the backup copy of the schemes file that was created to identify what has been changed.

AMQ4473
The WMQ_Schemes.xml file used to save schemes was found to be in an invalid format.

A backup copy of this file was made:

<insert_0>.

All user-defined schemes must be re-created.

Severity
10: Warning

Explanation
WebSphere MQ Explorer was unable to process the WMQ_Schemes.xml file as it had an invalid format. It was possibly truncated.

Response
Re-create all user-defined schemes. If possible, refer to the backup copy of the schemes file to obtain information.

AMQ4474
The WMQ_Filters.xml file used to save filters is incomplete. A backup copy of this file has been made: *<insert_0>*. Where possible, user-defined filters from this file have been extracted and retained, but it is possible that some have been lost.

Severity
10: Warning

Explanation
When reading in filters from the WMQ_Filters.xml file, some required information was missing.

Response
Re-create user-defined filters where necessary. Refer to the backup copy of the filters file that was created to identify what has been changed.

AMQ4475

The WMQ_Filters.xml file used to save filters was found to be in an invalid format. A backup copy of this file was made: <*insert_0*>. All user-defined filters must be re-created.

Severity

10: Warning

Explanation

WebSphere MQ Explorer was unable to process the WMQ_Filters.xml file as it had an invalid format. It was possibly truncated.

Response

Re-create all user-defined filters. If possible, refer to the backup copy of the filters file to obtain information.

AMQ4476

The WMQ_Sets.xml file used to save sets was found to be in an invalid format. A backup copy of this file was made: <*insert_0*>. All sets must be re-created.

Severity

10: Warning

Explanation

WebSphere MQ Explorer was unable to process the WMQ_Sets.xml file as it had an invalid format. It was possibly truncated.

Response

Re-create all sets as necessary. If possible, refer to the backup copy of the sets file that was created to obtain information.

AMQ4477

The topic string supplied is invalid.

Severity

10: Warning

Explanation

A topic string was missing or contained invalid characters.

Response

Ensure a topic string has been defined or that there are no invalid characters in the topic string.

AMQ4478

The publication could not be retained.

Severity

10: Warning

Explanation

An attempt was made to publish a message on a topic, using the MQPMO_RETAIN option, but the publication could not be retained. The publication was not published to any matching subscribers. Retained publications are stored on the SYSTEM.RETAINED.PUB.QUEUE. Possible reasons for failure include the queue being full, the queue being 'put' inhibited, or the queue not existing.

Response

Ensure that the SYSTEM.RETAINED.PUB.QUEUE queue is available for use by the application.

AMQ4479

An MQOPEN or MQPUT1 call was issued, specifying an alias queue as the target, but the BaseObjectName in the alias queue attributes was not recognized as a queue or topic name.

Severity

20: Error

Explanation
This error can also occur when BaseObjectName is the name of a cluster queue that cannot be resolved successfully.

Response
Correct the queue definitions.

AMQ4480
An MQOPEN or MQPUT1 call was issued, specifying an alias queue as the target, but the BaseObjectName in the alias queue definition resolves to a queue that is not a local queue, or local definition of a remote queue.

Severity
20: Error

Response
Correct the queue definitions.

AMQ4481
An error occurred when unsubscribing from the topic. The operation failed with reason code <*insert_0*>.

Severity
20: Error

Response
Use the reason code to determine the underlying reason for the failure.

AMQ4482
An error occurred when obtaining a publication. The operation failed with reason code <*insert_0*>.

Severity
20: Error

Explanation
An error occurred when performing a get operation for the subscribed topic. The topic was automatically unsubscribed.

Response
Use the reason code to determine the underlying reason for the failure.

AMQ4483
An error occurred when publishing a message on the topic. The operation failed with reason code <*insert_0*>.

Severity
20: Error

Response
Use the reason code to determine the underlying reason for the failure.

AMQ4484
An error occurred when obtaining the topic string for a publication. The operation failed with reason code <*insert_0*>.

Severity
20: Error

Explanation
The topic was automatically unsubscribed.

Response
Use the reason code to determine the underlying reason for the failure.

AMQ4485

This action removes the retained publication from the topic string <*insert_0*> on the selected queue manager only.

Are you sure you want to clear the retained publication?

Severity

10: Warning

Explanation

A confirmation is required before the retained publication is cleared.

Response

Continue only if you want to permanently clear the retained publication on this topic string.

AMQ4486

The retained publication on the topic string <*insert_0*> has been successfully cleared.

Severity

0: Information

Response

Message for information only.

AMQ4487

Error initialising <*insert_0*>.

Severity

30: Severe error

Explanation

An error occurred while starting this application.

Response

Check that the WebSphere MQ runtime libraries are available and the PATH system environment variable includes the directory for these runtime libraries.)

AMQ4488

Unable to locate a Web browser, product documentation, or IBM Eclipse Help System to display the help.

Severity

10: Warning

Explanation

To launch the help system, the Web browser or product documentation or IBM Eclipse Help System must be included in the PATH system environment variable.

Response

Install the product documentation or IBM Eclipse Help System or set the available Web browser on the system path. Re-launch the application and try again.

AMQ4489

Error launching the IBM Eclipse Help System.

Severity

10: Warning

Explanation

The application failed to create an instance of the IBM Eclipse Help System.

Response

Check that the IBM Eclipse Help System has been installed.

AMQ4490

Error starting the IBM Eclipse Help System.

Severity
10: Warning

Explanation
The application failed to start the IBM Eclipse Help System.

Response
Check that the IBM Eclipse Help System has been installed.

AMQ4491
Error launching the help system with a Web browser.

Severity
10: Warning

Explanation
The application failed to launch the help system through a Web browser.

Response
Check that the Web browser specified in the system path is working.

AMQ4492
Error launching the help system with IBM Eclipse Help System.

Severity
10: Warning

Explanation
The application failed to launch the help system through IBM Eclipse Help System.

Response
Check that the IBM Eclipse Help System has been installed.

AMQ4493
The help documentation is not available on the system.

Severity
10: Warning

Explanation
The application failed to locate the help documentation on the system.

Response
Check that the available help documentation for WebSphere MQ is installed.

AMQ4494
Unable to locate a Web browser in the system path.

Severity
10: Warning

Explanation
The application failed to locate a Web browsers in the system path.

Response
Check that a suitable Web browser is specified in the system path.

AMQ4495
This action resynchronizes all the proxy subscriptions with all other directly connected queue managers in all clusters and hierarchies in which this queue manager is participating.

Are you sure you want to continue with this action?

Severity
10: Warning

Explanation

This should only be used if the queue manager is receiving proxy subscriptions that it should not be, or is not receiving proxy subscriptions that it should be.

Missing proxy subscriptions can be observed if the closest matching Topic definition has been specified with Publication scope or Subscription scope set to Queue Manager, or if it has an empty or incorrect Cluster name.

Extraneous proxy subscriptions can be observed if the closest matching Topic definition has been specified with Proxy subscription behavior set to Force.

Response

Check the Topic definitions before resynchronizing the proxy subscriptions.

AMQ4496

The request to refresh the proxy subscriptions was accepted by WebSphere MQ.

Severity

0: Information

Response

Message for information only.

AMQ4497

The topic string has already been specified for another topic. Enter a different topic string.

Severity

10: Warning

Response

Enter a different topic string.

AMQ4498

This action removes the retained publication from the topic string <*insert_0*> on all queue managers connected in the Publish/Subscribe cluster.

Are you sure you want to clear the retained publication?

Severity

10: Warning

Explanation

A confirmation is required before the retained publication is cleared.

Response

Continue only if you want to permanently clear the retained publication on this topic string.

AMQ4499

The queue attribute for the JMS queue <*insert_0*> is empty. A queue name needs to be entered before mapping the JMS queue to an MQ queue.

Severity

10: Warning

Explanation

The user has not entered a queue name for the JMS Queue and therefore an MQ Queue cannot be created.

Response

Enter a value for the queue attribute on the JMS Queue and then try to create the MQ Queue again.

AMQ4500

Are you sure that you want to forcibly remove queue manager <*insert_0*> from cluster <*insert_1*>?

Severity
10: Warning

Explanation
You should only forcibly remove a queue manager from a cluster when it has already been deleted and cannot be removed from the cluster in the normal way. A confirmation is required before the queue manager is forcibly removed.

Response
Continue only if you want to forcibly remove the queue manager.

AMQ4501
The queue manager was successfully removed from the cluster. This might take some time to be reflected in the WebSphere MQ Explorer.

Severity
0: Information

Explanation
The queue manager will still appear as a member of the cluster until the configuration changes have been sent across the network and the cluster channels to the queue manager have become inactive. This might take a long time.

AMQ4502
You have shared the queue in cluster <*insert_0*>. The queue manager is not a member of this cluster.

Severity
10: Warning

Response
To make the queue available to the members of this cluster, you must join the queue manager to the cluster.

AMQ4503
The list of values is too long.

Severity
10: Warning

Explanation
The list of values that you have entered is too long. The maximum number of characters allowed for this value is <*insert_0*>.

AMQ4504
The value is too long.

Severity
10: Warning

Explanation
You have entered a value containing too many characters. The maximum number of characters allowed for each value of this attribute is <*insert_0*>.

AMQ4505
There are too many entries in the list.

Severity
10: Warning

Explanation
You have entered too many values in the list. The maximum number of values is <*insert_0*>.

AMQ4506

Cannot connect to queue manager <*insert_0*>. It cannot be removed from the cluster in the normal way.

Severity

10: Warning

Response

Try the operation again when the queue manager is available. If the queue manager no longer exists, you can choose to forcibly remove the queue manager from the cluster.

AMQ4507

The remote queue manager is not using TCP/IP.

Severity

10: Warning

Explanation

The connection information available for the remote queue manager uses a communication protocol other than TCP/IP. The WebSphere MQ Explorer cannot connect to the queue manager to remove it from the cluster in the normal way.

Response

If the queue manager no longer exists, you can choose to forcibly remove the queue manager from the cluster.

AMQ4508

The queue manager successfully left the cluster.

Severity

0: Information

Explanation

The queue manager will still appear as a member of the cluster until the configuration changes have been sent across the network and the cluster channels to the queue manager have become inactive. This might take a long time.

AMQ4509

The request to suspend membership of the cluster has been accepted.

Severity

0: Information

Response

Message for information only.

AMQ4510

The request to resume membership of the cluster has been accepted.

Severity

0: Information

Response

Message for information only.

AMQ4511

The queue manager is not a member of the cluster.

Severity

0: Information

Response

Message for information only.

AMQ4512
An error occurred while performing a cluster operation. The operation failed with error <*insert_0*>.

Severity
0: Information

Response
Message for information only.

AMQ4513
The request to refresh the information about the cluster has been accepted.

Severity
0: Information

Response
Message for information only.

AMQ4514
The queue manager is not a member of cluster <*insert_0*>.

Severity
10: Warning

Explanation
The object that you have shared in the cluster will not be available to other members of the cluster until you make this queue manager a member of the cluster.

AMQ4515
The repository queue manager for cluster <*insert_0*> is not available for connection.

Severity
10: Warning

Explanation
Views showing cluster queues in this cluster might not be complete.

AMQ4516
Cluster workload exit error.

Severity
10: Warning

Explanation
The queue manager's cluster workload exit failed unexpectedly or did not respond in time.

AMQ4517
Cluster resolution error.

Severity
10: Warning

Explanation
The definition of the cluster queue could not be resolved correctly because a response from a repository queue manager was not available.

AMQ4518
AMQ4518=A call was stopped by the cluster exit.

Severity
10: Warning

Explanation
The queue manager's cluster workload exit rejected a call to open or put a message onto a cluster queue.

AMQ4519
No destinations are available.

Severity
10: Warning

Explanation
At the time that the message was put, there were no longer any instances of the queue in the cluster.

AMQ4520
The WebSphere MQ Explorer could not initialize TCP/IP. Administration of remote queue managers and clusters is not possible.

Severity
10: Warning

AMQ4521
The text you entered contained a comma (,) which is used as a list separator character.

Severity
10: Warning

Explanation
This value does not accept lists.

Response
If you want to use a comma as part of a value, enclose the value in double quotation marks.

AMQ4522
The wizard was unable to add the queue manager to the cluster.

All changes will be rolled back.

Severity
10: Warning

Explanation
A problem occurred while defining objects or modifying the queue manager's properties.

Response
Ensure that the default objects exist for the queue manager.

AMQ4523
The wizard was unable to add one of the queue managers to the cluster.

All changes will be rolled back.

Severity
10: Warning

Explanation
A problem occurred while defining objects or modifying one of the queue managers' properties.

Response
Ensure that the default objects exist for the queue manager.

AMQ4524
The queue manager <*insert_0*> is the source repository in cluster <*insert_1*>. Removing this queue manager from the cluster prevents further repository actions from being run. To enable repository actions again, re-select another queue manager as the source of information. Are you sure that you want to remove this queue manager?

Severity
10: Warning

Explanation

To be able to display cluster information, the clustering component of the WebSphere MQ Explorer requires at least one full repository to be selected as the source. Removing the last full repository prevents the display of cluster members, and hence will prevent cluster actions being run on these full repositories.

Response

Select Yes if you want to remove the source repository, even though it will prevent access to remaining cluster information.

AMQ4525

Cluster workload exit load error.

Severity

10: Warning

Explanation

The queue manager's cluster workload exit failed to load.

Response

Check that the cluster workload exit exists and the name has been specified correctly.

AMQ4526

During the import further plug-ins were enabled. Do you want to import their settings?

Severity

0: Information

Explanation

The import file contains settings for the plug-ins enabled during the import.

Response

Select Yes to import the settings.

AMQ4527

Default Configuration is already running.

Severity

10: Warning

Explanation

There is an instance of default configuration already running on the system.

Response

Use the previously launched default configuration application. If you fail to get the previous default configuration dialog, stop the JVM running the application and try re-launching the application.

AMQ4528

The file selected does not contain any import settings.

Severity

20: Error

Response

Select another file and try again.

AMQ4529

Put message failed. The page set ID specified for the storage class defined for this queue is not valid.

Severity

20: Error

Explanation
The MQPUT or MQPUT1 call was issued, but the page set id specified in the storage class object defined for the queue is not valid.

Response
Correct the page set ID value in the storage class definition used by this queue and try again. If the error persists contact your System Administrator.

AMQ4530
The request to create and start a new z/OS listener was accepted.

Severity
0: Information

Explanation
A user request to create the listener was accepted by WebSphere MQ.

Response
Message for information only.

AMQ4531
The subscription is in use.

Severity
20: Error

Explanation
An attempt was made to delete or change a subscription in use.

Response
Ensure that the subscription is not in use and try again.

AMQ4547

Severity
20: Error

Explanation
Unable to load system libraries as the java.library.path and the native library path reference different installations.

Response
Ensure that the native library path (LD_LIBRARY_PATH, LIBPATH, or SHLIB_PATH) is set correctly.

AMQ4548

Severity
20: Error

Explanation
MQ Explorer encountered a problem with the system browser when trying to display the web page.

Response
Ensure that a browser is available to display the web page. If symptoms persist, contact your System Administrator.

AMQ4549
An unexpected error occurred copying settings from workspace <*insert_0*>.

Severity
10: Warning

Explanation
Some files or preferences could not be copied from the previous workspace.

Response
Ensure that the Eclipse workspace exists at the specified location and can be read.

AMQ4570
The requested application is either not installed or could not be launched.

Severity
20: Error

Response
Check that the corresponding product feature has been installed successfully. If symptoms persist contact your System Administrator.

AMQ4571
Are you sure that you want to change the location of the Key Repository for queue manager <*insert_0*>?

Severity
10: Warning

Explanation
You might prevent the queue manager from starting if you change the Key Repository field to a location which is not valid.

Response
Ensure that the location specified is correct before continuing.

AMQ4572
The request to refresh the information about all clusters has been accepted.

Severity
0: Information

Response
Message for information only.

AMQ4573
A queue manager has not been entered in the <*insert_0*> field on the <*insert_1*> page. A value must be entered in this field before the Select button can be used to set the <*insert_2*> field. Note that this value can also be entered manually.

Severity
20: Error

Explanation
WebSphere MQ Explorer needs to know exactly which queue manager to query to populate the object selection dialog.

Response
Enter a valid value into the appropriate field

AMQ4574
IBM WebSphere Explorer is already running.

Severity
30: Severe error

AMQ4575
An error occurred initializing the data model.

Severity
30: Severe error

AMQ4576
The working directory <*insert_0*> is not valid.

Severity
30: Severe error

AMQ4577
An error occurred initializing the process.

Severity
30: Severe error

AMQ4578
An error occurred loading the messages file <*insert_0*>.

Severity
30: Severe error

AMQ4579
An error occurred loading the system libraries.

Severity
30: Severe error

AMQ4580
An internal method detected an unexpected system return code. The method <*insert_0*> returned <*insert_1*>.

Severity
30: Severe error

Response
Examine the problem determination information on this computer to establish the cause of the error.

AMQ4581
Parameter check failed on the internal function <*insert_0*>. The error was <*insert_1*>.

Severity
30: Severe error

Response
Examine the problem determination information on this computer to establish the cause of the error.

AMQ4582
Queue manager <*insert_0*> is not available for client connection.

Severity
30: Severe error

Response
Ensure the queue manager is running and is configured to accept remote connections.

AMQ4583
Queue manager <*insert_0*> is not available for connection.

Severity
30: Severe error

Response
Ensure the queue manager is running.

AMQ4584
Queue manager <*insert_0*> is not available for cluster connection.

Severity
30: Severe error

Response
Ensure that the queue manager is running. If the queue manager has been deleted it might continue to be displayed as a member of a cluster for up to 30 days.

AMQ4585
An internal method <*insert_0*> encountered an unexpected error.

Severity
30: Severe error

Response
Examine the problem determination information on this computer to establish the cause of the error.

AMQ4586
The attempt to create the URL for file <*insert_0*> failed.

Severity
30: Severe error

Explanation
The file name specified was not recognized.

Response
Ensure that the file exists at the specified location and can be read.

AMQ4587
The attempt to read from URL <*insert_0*> failed.

Severity
30: Severe error

Explanation
There was an error when the system tried to read the Client channel definition table.

Response
Ensure that the file exists at the specified location and can be read.

AMQ4588
The attempt to read from URL <*insert_0*> failed.

Severity
30: Severe error

Explanation
There was an error when the system tried to read the file.

Response
Ensure that the file exists at the specified location and can be read.

AMQ4589
No connection was found to application <*insert_0*>.

Severity
10: Warning

Explanation
The connection was not found. Possibly the connection was closed before the command was issued.

Response
Check that the application connection has not been closed in the background.

AMQ4590
The queue manager connection to application <*insert_0*> could not be closed.

Severity
20: Error

Explanation
The connection could not be closed due to a PCF error.

Response
Check for FFSTs.

AMQ4591
The command server for <*insert_0*> is not running.

Severity
30: Severe error

Explanation
The command server has stopped for some reason, so the request cannot be processed.

Response
Start the command server. If the error persists, examine the problem determination information to see if any details have been recorded.

AMQ4592
The connection was closed successfully.

Severity
0: Information

Explanation
The request to close the connection to an application was successful.

Response
Message for information only.

AMQ4593
Do you really want to stop the connection to application <*insert_0*>

Severity
0: Information

Explanation
WebSphere MQ explorer is about to stop a connection, stopping the connection will prevent further communication between MQ and the application in question.

Response
Select yes if you want to stop the connection.

AMQ4594
The queue manager connection to application <*insert_0*> has not been closed.

Severity
0: Information

Explanation
Certain WebSphere MQ queue manager processes cannot be stopped.

Response
Message for information only.

AMQ4595
No response was received to the request to close the connection to application <*insert_0*>.

Severity
30: Severe error

Explanation
The command server might no longer be running.

Response
If the error persists, examine the problem determination information to see if any details have been recorded.

AMQ4596
Key store file <*insert_0*> cannot be found.

Severity
10: Warning

Explanation
The SSL key store or trust store does not exist.

Response
Create a new store file or change the connection property. Then try the request again.

AMQ4597
No certificates were loaded from key store file <*insert_0*>.

Severity
10: Warning

Explanation
The SSL key store or trust store does not contain any certificates.

Response
Add the appropriate certificates to the key store file. Then try the request again.

AMQ4598
Key store file <*insert_0*> could not be opened with the given password.

Severity
10: Warning

Explanation
The SSL key store or trust store could not be opened.

Response
Change the password. Then try the request again.

AMQ4599
Changing the FIPS required setting will affect all client connections using SSL and requires the WebSphere MQ Explorer to be restarted. Are you sure that you want to restart the WebSphere MQ Explorer now ?

Severity
10: Warning

Explanation
The FIPS required value is an application-wide setting and can only be changed from the Preferences page. All client connections using SSL will be affected by this setting.

Response
Restart the WebSphere MQ Explorer to apply this change.

AMQ4600
The password store <*insert_0*> could not be opened using the given key.

Severity
10: Warning

Explanation
The specified password store file cannot be opened.

Response
Make sure the password store file exists. Enter a different key and try again.

AMQ4601
Do you want to copy entries from the old password store to the new one?

Severity
10: Warning

Explanation
The user has changed the name of the password store file.

Response
Click Yes to copy entries to the new file.

AMQ4602
Unable to validate the given key for password store <*insert_0*>.

Severity
10: Warning

Explanation
The password store cannot be opened with the specified key.

Response
Enter a different key and try the operation again.

AMQ4603
Invalid password store <*insert_0*>.

Severity
10: Warning

Explanation
The file name is the name of a directory.

Response
Enter a valid file name.

AMQ4604
Password store <*insert_0*> is read-only.

Severity
10: Warning

Explanation
WebSphere MQ Explorer only has read access to the file name.

Response
Specify the name of a file that has both read and write access.

AMQ4605
Format of password store <*insert_0*> is unknown.

Severity
10: Warning

Explanation
The contents of the password store file is unknown. This may be an existing XML file which has not been created as a password store or a non-XML file.

Response
Specify an existing password store file name or specify a new XML file.

AMQ4606
Password store <*insert_0*> was not opened.

Severity
10: Warning

Explanation
The user chose not to open the password store.

Response
Restart the WebSphere MQ Explorer to open the password store or use the Password preference page.

AMQ4607
Queue manager has been disabled for Publish/Subscribe operations.

Severity
10: Warning

Explanation
An error occurred trying to perform a publish or subscribe operation.

Response
Change the PSMODE attribute on the queue manager to enable Publish/Subscribe operations.

AMQ4608
The specified destination does not exist.

Severity
30: Severe error

Explanation
An error occurred trying to create a new subscription.

Response
Change the destination name and try again.

AMQ4609
The listener was started.

Severity
0: Information

Explanation
The request to start a listener was successful.

Response
Message for information only.

AMQ4610
Invalid connection name.

Severity
10: Warning

Explanation
The connection name in the channel definition could not be resolved into a network address. Either the name server does not contain the entry, or the name server was not available.

Response
Ensure that the connection name is correctly specified and that the name server is available.

AMQ4611
Applying these changes will disconnect the queue manager and reconnect with the new details. Do you want to continue?

Severity
0: Information

Explanation
Connection details have been changed to a connected queue manager. Without reconnecting, the current connection details cannot be seen.

Response
Select yes to continue or no to cancel the changes.

AMQ4616
A newer command level has been found when connecting to <*insert_0*>. The old level is <*insert_1*> and the new level is <*insert_2*>. The connection to the queue manager will be replaced.

Severity
0: Information

Explanation
A previous connection to this queue manager has been successful; the queue manager is the same but the command level is now higher. The version of WebSphere MQ has been changed.

Response
Message for information only.

AMQ4620
Channel Authentication Record already exists.

Severity
20: Error

Explanation
An attempt was made to add a Channel Authentication Record, but it already exists.

Response
Use the properties panel to change an existing record.

AMQ4621
Channel Authentication Record not found.

Severity
20: Error

Explanation
The specified channel authentication record does not exist.

Response
Specify a channel authentication record that exists.

AMQ4622
A Channel Authentication Record contained an IP address with a range that conflicted with an existing range.

Severity
20: Error

Explanation
A range must be a complete superset or subset of any existing ranges for the same channel profile name.

Response
Specify a range that is a superset or a subset of existing ranges.

AMQ4623
The maximum number of Channel Authentication Records has been exceeded.

Severity
20: Error

Explanation
A Channel Authentication Record was set taking the total number of entries for that type on a single channel profile, over the maximum number allowed.

Response
Remove some Channel Authentication Records to make room.

AMQ4624
A Channel Authentication Record contained an invalid IP address.

Severity
20: Error

Explanation
A Channel Authentication Record contained an invalid IP address, or invalid wildcard pattern to match against IP addresses.

Response
Specify a valid IP address.

AMQ4625
A Channel Authentication Record contained an invalid IP address range.

Severity
20: Error

Explanation
A Channel Authentication Record contained an IP address with a range that was invalid, for example, the lower number is higher or equal to the upper number of the range.

Response
Specify a valid range in the IP address.

AMQ4626
The Channel Authentication Record client user value is not valid.

Severity
20: Error

Explanation
The client user value contains a wildcard character which is not allowed.

Response
Specify a valid value for the client user field.

AMQ4627
Channel authentication profile name is invalid.

Severity
20: Error

Explanation
The channel profile name used in the command was not valid. This might be because it contained characters which are not accepted names, or characters which are not valid for the specified profile type.

Response
Specify a valid value for the channel authentication profile name.

AMQ4700
PCF command identifier (<*insert_0*>) not valid for queue manager <*insert_1*>.

Severity
10: Warning

Explanation
The specified PCF command is not supported by this queue manager.

AMQ4701
The command level of the queue manager does not support the requested version of the command.

Severity
10: Warning

Explanation
There is a mismatch between the command requested and the command level supported by the queue manager. This might be because an intermediate queue manager is being used which is of a lower command level than the remote queue manager.

Response
Ensure that the intermediate queue manager is at the same or higher command level than the queue manager it is being used to connect to. If necessary, reconnect to the queue manager using a different intermediate queue manager.

AMQ4702
The current filter not supported for queue manager <*insert_0*>.

Severity
10: Warning

Explanation
The filter being applied to this view is not supported by this queue manager.

Response
Ensure that the filter settings are supported by the queue manager.

AMQ4766
Setup needs to install or upgrade this computer to version 2.0 of Microsoft Windows Installer. (MSI).

Explanation
After the upgrade you might need to reboot.

Response
Select Yes or No to Proceed.

AMQ4800
Error initializing <*insert_0*>.

Severity
30: Severe error

Explanation
An error occurred while starting this application.

Response
Check that the WebSphere MQ runtime libraries are available.

Check that the PATH system environment variable includes the directory for these runtime libraries.)

AMQ4807
The message size specified (<*insert_0*>) is outside the permitted range.

Severity
10: Warning

Response
Specify a value of 1000 to 100 000 000.

AMQ4808
Unknown <*insert_0*> <*insert_1*>.

Severity
10: Warning

Explanation
The named entity for the particular type is not defined on the system.

Response
Make sure the entity is defined and it matches the type of entity.

AMQ4809
You are about to delete the authority for <*insert_0*> to <*insert_1*>. Are you sure that you want to continue?

Severity
10: Warning

Explanation
You must confirm that you want to delete the specified authority. The entity name and object name are provided in the message.

Response
Continue only if you want to permanently delete the authority.

AMQ4810
The authority for <*insert_0*> to <*insert_1*> was deleted successfully.

Severity
0: Information

Response
Message for information only.

AMQ4811
The authority was created successfully.

Severity
0: Information

Response
Message for information only.

AMQ4812
You are about to delete all create authorities for <*insert_0*>. Are you sure that you want to continue?

Severity
10: Warning

Explanation
You must confirm that you want to delete the specified authority. The entity name is provided in the message.

Response
Continue only if you want to permanently delete the authority.

AMQ4813
You are about to refresh SSL security for <*insert_0*>. This might affect the running status of active channels. Are you sure that you want to continue?

Severity
10: Warning

Explanation
A confirmation is required before the refresh command is issued. Certain active channel types might be stopped as a result of this command. The queue manager name is provided in the message.

Response
Continue only if you want to refresh SSL security.

AMQ4814
The command server is not allowing security requests.

Severity
10: Warning

Explanation
The command server has been started with the "-a" option which blocks security related PCFs.

Response
Restart the command server without using the "-a" option.

AMQ4815
You are about to add authority for a non-generic profile name <*insert_0*>. Are you sure that you want to continue?

Severity
10: Warning

Explanation
You chose to add authorities for a generic profile name, but entered the name for a specific profile.

Response
Continue if you want to add authority for a specific profile name.

AMQ4816
The list of authorizations held internally by the authorization services component will be refreshed. Are you sure that you want to continue?

Severity
10: Warning

Explanation
A confirmation is required before the refresh command is issued.

Response
Continue only if you want to refresh authorization service component security.

AMQ4817
The in-storage profiles for the requested resources will be refreshed. Are you sure that you want to continue?

Severity
10: Warning

Explanation
A confirmation is required before the refresh command is issued to the WebSphere MQ in-storage ESM (External Security Manager).

Response
Continue only if you want to refresh the ESM.

AMQ4818
No authority records were found.

Severity
10: Warning

Explanation
There are no authority records matching the specific request.

Response
Change the entity or profile name and try again.

AMQ4819
Unable to write to file <*insert_0*>.

Severity
10: Warning

Explanation
You do not have write access to the file name.

Response
Check that your userid has write access to the file name.

AMQ4820
A file called <*insert_0*> already exists. Do you want to replace this file?

Severity
0: Information

Response
Confirm that you want to replace the file.

AMQ4821
This action replaces an existing authority record. Are you sure that you want to continue?

Severity
0: Information

Explanation
An explicit authority record already exists for this entity. Creating a new authority record replaces the existing authority record.

Response
Continue only if you want replace the existing authority record.

AMQ4822
You must enter a specific profile name when using an entity name.

Severity
0: Information

Response
Enter a specific profile name.

AMQ4823
Profile <*insert_0*> does not exist.

Severity
0: Information

Explanation
The profile name entered by the user does not exist for the type of object.

Response
Change the name of the profile or use the select button and try again.

AMQ4824
Invalid profile name <*insert_0*>.

Severity
0: Information

Explanation
The generic profile name entered by the user is not allowed.

Response
Change the name of the profile to match the supported wildcard characters and try again.

AMQ4825
The security exit class <*insert_0*> is invalid or cannot be found.

Severity
10: Warning

Response
Ensure that the security exit class is available and that it implements the com.ibm.mq.MQSecurityExit interface.

AMQ4826
There is a security profile case conflict.

Severity
10: Warning

Explanation
The security profile case attribute of the queue manager is different from that issued on the refresh command.

Response
Change the security profile case attribute of the queue manager or of the class specified on the refresh command.

AMQ4830
You are about to add authority for a generic profile name "<*insert_0*>". Are you sure that you want to continue?

Severity
10: Warning

Explanation
You chose to add authorities for a specific profile name, but entered the name for a generic profile.

Response
Continue if you want to add authority for a generic profile name.

AMQ4850
Further tests cannot be run because the WebSphere MQ Explorer Test Plug-in is currently in use.

Severity
10: Warning

Explanation
You must either cancel these tests or wait for them to complete before initiating further tests.

Response
Either stop the current tests using the progress view, or wait until the current tests are completed.

AMQ4851
There are no tests available to run.

Severity
0: Information

Explanation
The configuration used to launch these tests has no tests selected, this could be because no tests are selected, or there are no appropriate tests available.

Response
Try a difference configuration which has tests enabled, or try testing from a different point to ensure that there are appropriate tests available.

AMQ4852
WebSphere MQ Explorer Test Plug-in initialization error.

Severity
20: Error

Explanation
An error has occurred during initialization of the a Tests Plug-in. This might cause problems with running tests.

Response
Examine the problem determination information to see if any details have been recorded.

AMQ4853
The test cannot be disabled because no configurations currently have this test enabled.

Severity
0: Information

Response
No further action is required; the test is already disabled.

AMQ4854
Finished running <*insert_0*> tests.

Severity
0: Information

Explanation
The requested test run is complete, and the number of tests specified have been run. This message can be disabled from the Tests plug-in preferences.

Response
No further action is required; the test run has finished

AMQ4855
The test run was canceled.

Severity
0: Information

Explanation
The requested test run was canceled as the result of a user request. This message can be disabled from the Tests plug-in preferences.

Response
Message for information only.

AMQ4856
Are you sure that you want to clear the subscription named <*insert_0*>?

For a managed destination, messages already queued to the destination will be deleted.

Severity
10: Warning

Explanation
A confirmation is required before the subscription is cleared.

Response
Continue only if you want to clear the subscription.

AMQ4857
The Subscription was cleared.

Severity
0: Information

Explanation
The subscription was cleared to a well defined state. For a managed destination any messages already queued to the destination were deleted.

Response
Message for information only.

AMQ4858
A parameter change has been detected.

Severity
0: Information

Explanation
A parameter has been changed without using the WebSphere MQ Explorer.

Response
Refresh the WebSphere MQ Explorer view and try the operation again.

AMQ4859
The requested function is not available.

Severity
0: Information

Explanation
WebSphere MQ Explorer was not able to carry out the function requested.

Response
Try again. If symptoms persist contact your System Administrator.

AMQ4860
The queue manager is running in standby mode.

Severity
0: Information

Explanation
The queue manager has been started in standby mode.

AMQ4861
WebSphere MQ cannot stop the listener because the listener is already stopped.

Severity
10: Warning

AMQ4862
The default remote administration listener LISTENER.TCP could not be deleted.

Severity
10: Warning

Explanation
A problem has occurred trying to delete the listener.

Response
Check that the listener has been stopped or that it has not already been deleted.

AMQ4863
The property <*insert_0*> has not been prefixed correctly.

Severity
20: Error

Explanation
Service definition destination names must be prefixed with 'msg/queue/' for queues, or 'msg/topic/' for topics.

Response
Prefix the destination name with the relevant prefix.

AMQ4864
The property <*insert_0*> is not the correct length.

Severity
20: Error

Explanation
Queue names cannot exceed 48 characters.

Response
Check that the name of the queue is correct.

AMQ4865
The property <*insert_0*> does not contain a destination name.

Severity
20: Error

Explanation
The value entered does not include the name of a destination.

Response
Enter the name of a valid destination. Service definition destination names must be prefixed with 'msg/queue/' for queues, or 'msg/topic/' for topics.

AMQ4866
The property <*insert_0*> is not a valid URI format.

Severity
20: Error

Explanation
Only valid URIs can be specified for this property.

Response
Check that the value entered is in a valid URI syntax.

AMQ4867
The property <*insert_0*> on page <*insert_1*> is not a valid URI format.

Severity
20: Error

Explanation
Only valid URIs can be specified for this property.

Response
Check that the value entered is in a valid URI syntax.

AMQ4868
An unexpected error has occurred.

Severity
20: Error

Explanation
An unexpected error has occurred.

Response
Contact your system administrator.

AMQ4869
The export location <*insert_0*> already exists. Do you want to overwrite the existing files?

Severity
10: Warning

Explanation
The export location already exists. If you continue, existing files may be overwritten.

Response
Confirm that you want you overwrite files at the chosen export location.

AMQ4870
Could not establish a connection to the queue manager. Channel not available.

Severity
10: Warning

Explanation
The attempt to connect to the queue manager failed. See reason code MQRC_CHANNEL_NOT_AVAILABLE for more information.

Response
Examine the queue manager and client error logs for messages explaining the cause of the problem.

AMQ4871
Could not establish a connection to the queue manager. Channel name not recognized.

Severity
10: Warning

Explanation
The attempt to connect to the queue manager failed. The queue manager did not recognize the channel name.

Response
Use a different channel name and try again.

AMQ4999
An unexpected error (<*insert_0*>) has occurred.

Severity
10: Warning

Explanation
An unlisted error has occurred in the system while retrieving PCF data.

Response
Try the operation again. If the error persists, examine the problem determination information to see if any details have been recorded.

AMQ5000-5999: Installable services

AMQ5005
Unexpected error

Severity
20 : Error

Explanation
An unexpected error occurred in an internal function of the product.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5006
Unexpected error: rc = <*insert_1*>

Severity
20 : Error

Explanation
An unexpected error occurred in an internal function of the product.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5008
An essential IBM WebSphere MQ process <*insert_1*> (<*insert_3*>) cannot be found and is assumed to be terminated.

Severity
40 : Stop Error

Explanation
1) A user has inadvertently terminated the process. 2) The system is low on resources. Some operating systems terminate processes to free resources. If your system is low on resources, it is possible it has terminated the process so that a new process can be created.

Response
IBM WebSphere MQ will stop all MQ processes. Inform your systems administrator. When the problem is rectified IBM WebSphere MQ can be restarted.

AMQ5009
IBM WebSphere MQ agent process <*insert_1*> has terminated unexpectedly.

Severity
40 : Stop Error

Explanation
IBM WebSphere MQ has detected that an agent process has terminated unexpectedly. The queue manager connection(s) that this process is responsible for will be broken.

Response
Try to eliminate the following reasons before taking any further action:

1) A user has inadvertently terminated the process.

2) The system is low on resources. Some operating systems terminate processes to free resources. If your system is low on resources, it is possible that the operating system has terminated the process so that a new process can be created. If you believe the problem is not a result of the above reasons, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at

http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5010
The system is restarting the WorkLoad Management Server process.

Severity
10 : Warning

Explanation
The system has detected that the WorkLoad Management server process (amqzlwa0, pid:<*insert_1*>) has stopped and is restarting it.

Response
Save the generated output files which may indicate the reason why the WorkLoad Management process stopped. If the reason the WorkLoad Management Server process stopped is a problem in a WorkLoad Management user exit, correct the problem, otherwise use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5011
The Queue Manager ended for reason <*insert_1*> <*insert_3*>

Severity
10 : Warning

Explanation
The Queue Manager ended because of a previous error <*insert_1*> or <*insert_3*>

Response
This message should be preceded by a message or FFST information from the internal routine that detected the error. Take the action associated with the earlier error information.

AMQ5019
Unable to access program <*insert_3*>.

Severity
40 : Stop Error

Explanation
A request was made to execute the program <*insert_3*>, however the operation was unsuccessful because the program could not be found in the specified location.

Response
Check the definition of the service specifies the correct and full path to the program to run. If the path is correct then verify that the program exists in the specified location and that WebSphere MQ userid has permission to access it.

AMQ5020
Permission denied attempting to execute program <*insert_3*>.

Severity
40 : Stop Error

Explanation
A request was made to execute the program <*insert_3*>, however the operation was unsuccessful because the IBM WebSphere MQ operating environment has insufficient permissions to access the program file.

Response
Check the access permissions of the of the program to be executed and if necessary alter them to

include execute permission for the IBM WebSphere MQ userId. Also check that the IBM WebSphere MQ userId has search access on all directories which compose the path to the program file.

AMQ5021
Unable to start program <*insert_3*>.

Severity
40 : Stop Error

Explanation
A request was made to execute the program <*insert_3*> however the operation was unsuccessful. Reasons for the failure may include

a shortage of available system resources

a problem with the program to be started

Response
If the problem persists then the IBM WebSphere MQ error logs should be consulted for further information related to this error. The Operating System error recording facilities should also be consulted for information relating to shortage of system resources.

AMQ5022
The Channel Initiator has started. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Channel Initiator process has started.

Response
None.

AMQ5023
The Channel Initiator has ended. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Channel Initiator process has ended.

Response
None.

AMQ5024
The Command Server has started. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Command Server process has started.

Response
None.

AMQ5025
The Command Server has ended. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Command Server process has ended.

Response
None.

AMQ5026
The Listener <*insert_3*> has started. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Listener process has started.

Response
None.

AMQ5027
The Listener <*insert_3*> has ended. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Listener process has ended.

Response
None.

AMQ5028
The Server <*insert_3*> has started. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Server process has started.

Response
None.

AMQ5029
The Server <*insert_3*> has ended. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Server process has ended.

Response
None.

AMQ5030
The Command <*insert_3*> has started. ProcessId(<*insert_1*>).

Severity
0 : Information

Explanation
The Command has started.

Response
None.

AMQ5032
Error (<*insert_4*>) accessing file <*insert_3*>.

Severity
40 : Stop Error

Explanation
While attempting to access the file <*insert_3*> the error <*insert_4*> occurred.

Response
Use the information contained in the error to locate and correct the cause of the failure.

AMQ5036
Error detected processing line <*insert_1*>, position <*insert_2*> in service environment file.

Severity
40 : Stop Error

Explanation
While processing the environment file <*insert_3*> an error was detected on line <*insert_1*> at position <*insert_2*>. Possible causes are

Variable name too long

Variable value too long

Incorrectly formed line. Lines must be in the format <name>=<value>. There should be no blank characters in name field. All characters following the '=' are part of the value field.

Response
This error will not stop the command from executing but any data on the invalid line is not processed.

AMQ5037
The Queue Manager task <*insert_3*> has started.

Severity
0 : Information

Explanation
The <*insert_4*> Utility Task Manager, processId(<*insert_1*>), has started the <*insert_3*> task. This task has now started <*insert_2*> times.

Response
None.

AMQ5038
The Queue Manager task <*insert_3*> failed to start with error-code <*insert_1*>.

Severity
40 : Stop Error

Explanation
The Utility Task Manager, attempted to start the task <*insert_3*> but the start request failed with error code <*insert_1*>.

Response
The failure to start the identified task may not be critical to queue-manager operation however all of the queue manager functionality may not be available. Further details of the failure are available in IBM WebSphere MQ error logs.

AMQ5041
The Queue Manager task <*insert_3*> has ended.

Severity
0 : Information

Explanation
The Queue Manager task <*insert_3*> has ended.

Response
None.

AMQ5042
Request to start <*insert_3*> failed.

Severity
40 : Stop Error

Explanation
The request to start the process <*insert_3*> failed.

Response
Consult the Queue Manager error logs for further details on the cause of the failure.

AMQ5043
Statistics recording is unavailable due to error code <*insert_1*>.

Severity
40 : Stop Error

Explanation
The statistics collection task was unable to start due the error code <*insert_1*>. Statistics collection will be unavailable until the problem is rectified and the Queue Manager is restarted.

Response
Consult the Queue Manager error logs for further details on the cause of the failure.

AMQ5044
<*insert_3*> task operation restricted due to Reason Code <*insert_1*>.

Severity
10 : Warning

Explanation
The <*insert_3*> task encountered a non-fatal error which may effect the operation of the task.

Response
Using the Reason Code <*insert_1*> and any previous messages recorded in the Error Logs correct the error. It may be necessary to restart the Queue Manager in order remove the restriction caused by the failure.

AMQ5045
System reconfiguration event received

Severity
0 : Information

Explanation
The Queue Manager received a system re-configuration event. This is likely to have been caused by an administrative change in the configuration of the machine (for example dynamically adding or removing resources such as memory or processors).

Response
No action is required unless this notification was unexpected.

AMQ5046
Automatic unmarking of messages is unavailable due to error code <*insert_1*>.

Severity
40 : Stop Error

Explanation
An error was encountered by the task that unmarks messages which have been marked for cooperative browse but have not been destructively got within the timeout period. The error code was <*insert_1*>. Automatic unmarking of messages will be unavailable until the problem is rectified and the queue manager is restarted.

Response
Consult the queue manager error logs for further details on the cause of the failure.

AMQ5049
The Queued Pubsub Daemon cannot be started/stopped due to error code <*insert_1*>.

Severity
40 : Stop Error

Explanation
An error was encountered by the task that starts and stops the queued pubsub daemon. The error code was <*insert_1*>. The daemon will be unable to be started or stopped until the problem is rectified and the queue manager is restarted.

Response
Consult the queue manager error logs for further details on the cause of the failure.

AMQ5050
An essential WebSphere MQ process <*insert_1*> *(*<*insert_3*>*)* cannot be found and is assumed to be terminated.

Severity
40 : Stop Error

Explanation
1) A user has inadvertently terminated the process. 2) The system is low on resources. Some operating systems terminate processes to free resources. If your system is low on resources, it is possible it has terminated the process so that a new process can be created. 3) MQ has encountered an unexpected error. Check for possible errors reported in the MQ error logs and for any FFSTs that have been generated.

Response
WebSphere MQ will attempt to restart the terminated process.

AMQ5051
The queue manager task <*insert_3*> has started.

Severity
0 : Information

Explanation
The critical utility task manager has started the <*insert_3*> task. This task has now started <*insert_2*> times.

Response
None.

AMQ5052
The queue manager task <*insert_3*> has started.

Severity
0 : Information

Explanation
The publish/subscribe utility task manager has started the <*insert_3*> task. This task has now started <*insert_2*> times.

Response
None.

AMQ5053
WebSphere MQ process <*insert_1*> (<*insert_3*>) cannot be found and is assumed to be terminated.

Severity
10 : Warning

Explanation
A queue manager process has terminated, the queue manager will continue to run but the functionality of the queue manager may be limited until the problem is resolved. Possible reasons for the termination are: 1) A user has inadvertently terminated the process. 2) The system is low on resources. Some operating systems terminate processes to free resources. 3) The process encountered an error.

Response
Check for earlier messages in the queue manager and system error logs that may indicate the problem. When the problem is rectified the queue manager will need to be restarted to restore the lost functionality.

AMQ5203
An error occurred calling the XA interface.

Severity
0 : Information

Explanation
The error number is <*insert_2*> where a value of

1 indicates the supplied flags value of <*insert_1*> was invalid,

2 indicates that there was an attempt to use threaded and non-threaded libraries in the same process,

3 indicates that there was an error with the supplied queue manager name <*insert_3*>,

4 indicates that the resource manager id of <*insert_1*> was invalid,

5 indicates that an attempt was made to use a second queue manager called <*insert_3*> when another queue manager was already connected,

6 indicates that the Transaction Manager has been called when the application is not connected to a queue manager,

7 indicates that the XA call was made while another call was in progress,

8 indicates that the xa_info string <*insert_3*> in the xa_open call contained an invalid parameter value for parameter name <*insert_4*>,

9 indicates that the xa_info string <*insert_3*> in the xa_open call is missing a required parameter, parameter name <*insert_4*>, and

10 indicates that MQ was called in dynamic registration mode but cannot find the ax_reg and ax_unreg functions ! Either call MQ in non-dynamic registration mode or supply the correct library name via the AXLIB parameter in the xa_open string.

Response
Correct the error and try the operation again.

AMQ5204
A non-threaded application tried to run as a Trusted application.

Severity
10 : Warning

Explanation
Only applications linked with the threaded MQ libraries can run as Trusted applications.

Response
Make sure that the application is relinked with the threaded MQ libraries, or set the environment variable MQ_CONNECT_TYPE to STANDARD.

AMQ5205
File or directory <*insert_3*> not owned by user <*insert_4*>.

Severity
10 : Warning

Explanation
IBM WebSphere MQ has detected that the file or directory <*insert_3*> is not owned by the user <*insert_4*>. This is not necessarily an error but you should investigate further if this is unexpected.

Response

If this is unexpected then you should alter the ownership of the file or directory back to the user <*insert_4*>.

If this is expected, then IBM WebSphere MQ will continue however WebSphere MQ will be unable to verify the security of this file or directory. If the access permissions are too strict then you may encounter problems if IBM WebSphere MQ cannot access the contents of the file or directory. If the access permissions are too relaxed then there may be an increased risk to the security of the IBM WebSphere MQ system.

AMQ5206
Duplicate parameters detected.

Severity
10 : Warning

Explanation
IBM WebSphere MQ has detected that the activity about to be displayed contains two or more parameters in the same group with the same parameter identifier. The activity may be displayed incorrectly.

Response
Inform the author of the activity that there may be an error in it.

AMQ5211
Maximum property name length exceeded.

Severity
10 : Warning

Explanation
IBM WebSphere MQ was in the process of parsing an MQRFH2 folder that is known to contain message properties. However, one of the elements in folder <*insert_3*> has a name which is longer than MQ_MAX_PROPERTY_NAME_LENGTH. The element name begins <*insert_4*>. The name of the parsed message property will be limited to the maximum number of characters which may cause inquiry of that property or selection of the message to fail.

Response
Reduce the size of the MQRFH2 element name or move the element into a folder which does not contain properties.

AMQ5358
IBM WebSphere MQ could not load AX support module <*insert_3*>.

Severity
20 : Error

Explanation
An error has occurred loading the AX support module <*insert_3*>. This module needs to be loaded so that dynamically-registering resource managers, such as Db2, can participate in global units of work.

Response
Look for a previous message outlining the reason for the load failure. Message AMQ6175 should have been issued if the load failed because of a system error. If this is the case then follow the guidance given in message AMQ6175 to resolve the problem. In the absence of prior messages or FFST information related to this problem check that the AX support module and the mqmax library have been correctly installed on your system.

AMQ5370
IBM WebSphere MQ client for HP Integrity NonStop Server (<*insert_1*>) enlisting with wrong TMF/Gateway.

Severity
10 : Warning

Explanation
A IBM WebSphere MQ client for HP Integrity NonStop Server, process (<*insert_1*>), connected to <*insert_3*> has incorrectly attempted to enlist with TMF/Gateway connected to <*insert_4*>.

Response
The configuration for the IBM WebSphere MQ client for HP Integrity NonStop Server is incorrect. Ensure the mqclient.ini TMF and TMFGateway stanza have been correctly configured to match the correct TMF/Gateway instances for the queue managers being used.

AMQ5371
TMF/Gateway shutting down due to TMF operator closing RM file <*insert_3*>.

Severity
20 : Error

Explanation
The TMF/Gateway is shutting down due to the TMF operator closing RM file <*insert_3*>.

Response
Contact the TMF administrator to determine why the RM file has been closed.

AMQ5372
TMF has shutdown.

Severity
10 : Warning

Explanation
TMF has shutdown. The TMF/Gateway for queue manager <*insert_3*> will reset and wait for TMF to become available before restarting operation.

Response
Contact the TMF administrator to determine why TMF has been shutdown.

AMQ5373
TMF not configured.

Severity
20 : Error

Explanation

The TMF/Gateway for queue manager <*insert_3*>is unable to start due to the TMF subsystem not being configured.

Response

Contact the TMF administrator to ensure the TMF subsystem is configured.

AMQ5374

TMF/Gateway not authorized to access RM file.

Severity

20 : Error

Explanation

The TMF/Gateway for queue manager <*insert_3*>is not authorized to access TMF RM file.

Response

There is an existing RM file <*insert_4*> within TMF, associated with a different owner from that specified for the TMF/Gateway server class for queue manager <*insert_3*> within Pathway.

Ensure the TMF/Gateway server class within Pathway is configured with the same owner as the existing TMF RM file.

AMQ5375

TMF/Gateway for queue manager <*insert_3*> has encountered a TMF resource error <*insert_1*>.

Severity

20 : Error

Explanation

The TMF/Gateway for queue manager <*insert_3*> has encountered a TMF resource error <*insert_1*>.

Response

These errors are typically as a result of reaching configured resource limits within the TMF subsystem. Refer to the HP NonStop Guardian Procedure Errors and Messages Manual for the appropriate corrective action based on the error <*insert_1*>.

AMQ5376

IBM WebSphere MQ

Severity

0 : Information

Explanation

Queue manager <*insert_3*> is unavailable for communication with the TMF/Gateway.

Response

Ensure that the queue manager has been started. The TMF/Gateway uses a client channel connection so additional channel definition and channel status checks might be required.

The TMF/Gateway will periodically attempt to reestablish communication with the queue manager.

If the queue manager continues to remain unavailable, this message will be reissued at regular intervals.

AMQ5377

The TMF/Gateway is not authorized to connect to queue manager <*insert_3*>.

Severity

20 : Error

Explanation
The TMF/Gateway is not authorized to connect to queue manager <*insert_3*>.

Response
Ensure the TMF/Gateway has been configured to use the correct queue manager and that queue manager has granted appropriate authority for the owner of the TMF/Gateway.

AMQ5378
Participation in TMF transactions is not support by queue manager <*insert_3*>.

Severity
20 : Error

Explanation
TMF/Gateway has detected WebSphere MQ for z/OS queue manager <*insert_3*> does not support participation in TMF transactions.

Response
The version of z/OS queue manager that you are connecting to does not support the TMF Gateway, please upgrade to a supported release.

AMQ5379
TMF/Gateway started with missing or invalid parameters

Severity
0 : Information

Explanation
Usage: runmqtmf -m QMgrName [-c ChannelName] [-h HostName] [-p Port] [-n MaxThreads] where:

-m is the name of the queue manager for this Gateway process. If you are using a queue sharing group (or other port distribution technology), this parameter must be targeted to a specific queue manager. This parameter is mandatory.

-c is the name of the server channel on the queue manager to be used by this gateway process. This parameter is optional.

-p is the TCP/IP port for the queue manager. This parameter is optional.

-h is the host name of the queue manager. This parameter is optional.

-n is the maximum number of worker threads that are created by the Gateway process. This parameter can be a value of 10 or greater. This parameter is optional. If no value is provided, the Gateway process creates up to a maximum of 50 threads.

If you specify one or more, but not all of the -c, -p, and -h attributes, then those attributes that you do not specify default to the following values:

ChannelName defaults to SYSTEM.DEF.SVRCONN

HostName defaults to localhost

Port defaults to 1414

Response
Ensure the TMF/Gateway is started with only valid parameters.

AMQ5380
A single TMF/Gateway process must be configured with TMF for each queue manager that is to participate in TMF coordinated units of work.

Severity
20 : Error

Explanation
None.

Response
Use the TMFCOM **STATUS RESOURCEMANAGER** command to identify the process that is already using RM-file <*insert_4*>.

If you are using multiple installations, you must nominate a single Gateway process from one of these installation to coordinate queue manager <*insert_3*>. The interface to the Gateway process supports any client at the same version or earlier. Ensure the TMF/Gateway server class definition within pathway for queue manager <*insert_3*> has been configured with MAXSERVER set to 1.

AMQ5390
Invalid process name <*insert_3*> provided in the MQTMF_GATEWAY_NAME environment variable for the TMF/Gateway for queue manager <*insert_4*>.

Severity
20 : Error

Explanation
Invalid process name <*insert_3*> provided in the MQTMF_GATEWAY_NAME environment variable for the TMF/Gateway for queue manager <*insert_4*>.

Response
Ensure the TMF/Gateway is running and the MQTMF_GATEWAY_NAME environment variable is correctly set to the Guardian process name of the TMF/Gateway.

AMQ5391
No PATHMON process name provided in the mqclient.ini for the TMF/Gateway for queue manager <*insert_3*>.

Severity
20 : Error

Explanation
None.

Response

Ensure an mqclient.ini file is available for use by the IBM WebSphere MQ client for HP Integrity NonStop Server and that it contains a TMFGateway stanza providing the server class name to be used for queue manager <*insert_3*>.

Refer to the IBM WebSphere MQ product documentation for further information about using an mqclient.ini file with the IBM WebSphere MQ client for HP Integrity NonStop Server system.

AMQ5392
No server class name provided in the mqclient.ini for the TMF/Gateway for queue manager <*insert_3*>.

Severity
20 : Error

Explanation
None.

Response

Ensure an mqclient.ini file is available containing a TMF stanza providing the Guardian process name of a PATHCOM that is hosting a TMF/Gateway server class for queue manager <*insert_3*>.

The mqclient.ini file also requires a TMFGateway stanza providing the server class name to be used for queue manager <*insert_3*>.

Refer to the IBM WebSphere MQ product documentation for further information about using an mqclient.ini file.

AMQ5393
The TMF/Gateway for queue manager <*insert_3*> is unable to process the request, return code (<*insert_1*>:<*insert_3*>).

Severity
20 : Error

Explanation
None.

Response
Check the TMF/Gateway error logs for further details.

AMQ5394
The TMF/Gateway for queue manager <*insert_3*> has successfully processed the request.

Severity
0 : Information

Explanation
None.

Response
None.

AMQ5395
Unable to locate server class <*insert_4*> hosted by PATHMON process <*insert_3*>.

Severity
20 : Error

Explanation
None.

Response

The configuration error may be one of the following:

1. The mqclient.ini TMFGateway stanza contains an invalid server class name for queue manager <*insert_5*>.
2. The PATHMON process <*insert_3*> has not been configured with server class <*insert_4*>.
3. Server class <*insert_4*> has not been started or is currently frozen.

AMQ5396
Unable to locate PATHMON process <*insert_3*>.

Severity
20 : Error

Explanation
None.

Response

The configuration error may be one of the following:

1. The mqclient.ini TMF stanza contains an invalid process name.
2. The PATHMON process <*insert_3*> is not currently running.

AMQ5397
Not authorized to use server class <*insert_4*> hosted by PATHMON process <*insert_3*>

Severity
20 : Error

Explanation
None.

Response
Check with your systems administrator to ensure you have the correct access permissions. When confirmed you have the correct access permissions, retry the operation.

AMQ5398
Error encountered while establishing contact with the TMF/Gateway server class <*insert_4*> hosted by PATHMON process <*insert_3*>. Pathsend error (<*insert_1*>), file system error (<*insert_2*>).

Severity
20 : Error

Explanation
None.

Response
These errors are typically the result of configuration problems with the PATHMON process <*insert_3*> or the server class <*insert_4*>. Refer to the HP NonStop TS/MP Pathsend and Server Programming Manual for the appropriate corrective action based on the pathsend error (<*insert_1*>) and file system error (<*insert_2*>).

AMQ5399
The TMF/Gateway server class <*insert_4*> hosted by PATHMON process <*insert_3*> has not be configured appropriately.

Severity
20 : Error

Explanation
None.

Response

The configuration error may be one of the following:

1. The server class has not been configured with TMF enabled.
2. The server class has been configured with MAXLINKS set too low for the number of IBM WebSphere MQ client for HP Integrity NonStop Server applications needing to concurrently enlist with the TMF/Gateway.

AMQ5501
There was not enough storage to satisfy the request

Severity
20 : Error

Explanation
An internal function of the product attempted to obtain storage, but there was none available.

Response
Stop the product and restart it. If this does not resolve the problem, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5502
The CDS directory name <*insert_3*> is not in the correct format.

Severity
20 : Error

Explanation

An internal function of the DCE Naming service found a CDS directory name in the wrong format. The name was expected to start with either '/...' for a fully qualified name (from global root), or '/.:' for a partially qualified name (from local cell root).

Response

Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5503

The name of the local DCE cell cannot be determined, status = <*insert_1*>

Severity

20 : Error

Explanation

The DCE Naming Service attempted to determine the name of the local DCE cell by calling 'dce_cf_get_cell_name()', which returned a nonzero return code.

Response

Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5504

DCE error. No value for the XDS attribute found.

Severity

20 : Error

Explanation

The DCE Naming service called om_get() to get the entry from the object returned by ds_read(). Although the status was correct, no objects were returned.

Response

Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5505

DCE error. No value for the XDS attribute number <*insert_1*> found.

Severity

20 : Error

Explanation

The DCE Naming service called om_get() to get the entry from the object returned by ds_read(). Although the status was correct, no objects were returned.

Response

Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5506

DCE error. <*insert_3*> returned <*insert_1*> for attribute number <*insert_2*>.

Severity
20 : Error

Explanation
The DCE Naming service queried an object by calling <*insert_3*> which returned a nonzero return code.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5507
DCE error. <*insert_3*> failed for an unknown reason.

Severity
20 : Error

Explanation
An unexpected error occurred in an internal function of the DCE Naming service.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5508
DCE error. The requested attribute is not present.

Severity
20 : Error

Explanation
The DCE Naming service was attempting to extract the value from an attribute, but the attribute cannot be found in the XDS object.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5509
DCE error. The XDS workspace cannot be initialized.

Severity
20 : Error

Explanation
The DCE Naming service called 'ds_initialize()' to initialize the XDS workspace, but 'ds_initialize()' returned a nonzero return code.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5510
DCE error. <*insert_3*> returned with problem <*insert_1*>.

Severity
20 : Error

Explanation
The DCE Naming service found an unexpected XDS error.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5511
Installable service component <*insert_3*> returned <*insert_4*>.

Severity
20 : Error

Explanation
The internal function, that adds a component to a service, called the component initialization process. This process returned an error.

Response
Check the component was installed correctly. If it was, and the component was supplied by IBM, then save the generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5511 (IBM i)
An installable service component returned an error.

Severity
20 : Error

Explanation
Installable service component <*insert_3*> returned <*insert_4*>. The internal function, that adds a component to a service, called the component initialization process. This process returned an error.

Response
Check the component was installed correctly. If it was, and the component was supplied by IBM, then save the generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5512
Installable service component <*insert_3*> returned <*insert_4*> for queue manager name = <*insert_5*>.

Severity
20 : Error

Explanation
An installable service component returned an unexpected return code.

Response
Check the component was installed correctly. If it was, and the component was supplied by IBM,

then save the generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5512 (IBM i)
An installable service component returned an unexpected return code.

Severity
20 : Error

Explanation
Installable service component <*insert_3*> returned <*insert_4*> for queue manager name = <*insert_5*>.

Response
Check the component was installed correctly. If it was, and the component was supplied by IBM, then save the generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5513
<*insert_3*> returned <*insert_1*>.

Severity
20 : Error

Explanation
An unexpected error occurred.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5519
Bad DCE identity. Status = <*insert_1*>, auth = <*insert_2*>, keytab file = <*insert_3*>, principal = <*insert_4*>.

Severity
20 : Error

Explanation
The keytab file was not installed correctly, or the WebSphere MQ user ID has a different password from that used to create the keytab file.

Response
Make sure that the MQ user ID defined when the product was installed has the same password as that defined by the keytab file, and that the keytab file has been installed correctly.

AMQ5519 (IBM i)
Bad DCE identity.

Severity
20 : Error

Explanation

Status = <*insert_1*>, auth = <*insert_2*>, keytab file = <*insert_3*>, principal = <*insert_4*>. The keytab file was not installed correctly, or the IBM WebSphere MQ user ID has a different password from that used to create the keytab file.

Response

Make sure that the MQ user ID defined when the product was installed has the same password as that defined by the keytab file, and that the keytab file has been installed correctly.

AMQ5520

The system could not load the module <*insert_5*> for the installable service <*insert_3*> component <*insert_4*>. The system return code was <*insert_1*>. The Queue Manager is continuing without this component.

Severity

10 : Warning

Explanation

The queue manager configuration data included a stanza for the installable service <*insert_3*> component <*insert_4*> with the module <*insert_5*>. The system returned <*insert_1*> when it tried to load this module. The Queue Manager is continuing without this component.

Response

Make sure that the module can be loaded. Put the module into a directory where the system can load it, and specify its full path and name in the configuration data . Then stop and restart the queue manager.

AMQ5520 (IBM i)

The system could not load a module. The Queue Manager is continuing without this component.

Severity

10 : Warning

Explanation

The queue manager configuration data included a stanza for the installable service <*insert_3*> component <*insert_4*> with the module <*insert_5*>. The system returned <*insert_1*> when it tried to load this module. The Queue Manager is continuing without this component.

Response

Make sure that the module can be loaded. Put the module into a directory where the system can load it, and specify its full path and name in the configuration data . Then stop and restart the queue manager.

AMQ5521

The system could not open "<*insert_3*>".

Severity

10 : Warning

Explanation

The system failed to open the default object "<*insert_3*>" at connect time for reason <*insert_4*>. This may be because "<*insert_3*>" has been deleted or changed.

Response

re-create the default objects by running "strmqm -c <qmgr>" (where <qmgr> is the name of the queue manager) and retry the application.

AMQ5522

A IBM WebSphere MQ installable service component could not be initialized.

Severity

20 : Error

Explanation
An installable service component returned an unexpected return code.

Response
Check the queue manager error logs for messages explaining which installable service could not be initialized and why that service could not be initialized. Check the component was installed correctly. If it was, and the component was supplied by IBM, then save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. If the component was not supplied by IBM, save the generated output files and follow the support procedure for that component.

AMQ5524
The IBM WebSphere MQ Object Authority Manager has failed to migrate authority data.

Severity
20 : Error

Explanation
The object authority manager has attempted to migrate existing queue manager authority data from a previous version of an object authority manager and failed.

Response
Check this log for any previous related messages, follow their recommendations then restart the queue manager.

AMQ5525
The IBM WebSphere MQ Object Authority Manager has failed.

Severity
20 : Error

Explanation
The object authority manager has failed to complete an MQ request.

Response
Check the queue manager error logs for messages explaining the failure and try to correct the problem accordingly.

AMQ5526
The IBM WebSphere MQ Object Authority Manager has failed with reason <*insert_1*>

Severity
20 : Error

Explanation
The object authority manager has failed an operation on the object authority manager's data queue <*insert_3*> with reason <*insert_1*>.

Response
Investigate why the error has occurred and correct the problem.

AMQ5527
The IBM WebSphere MQ Object Authority Manager has failed to locate an essential authority file

Severity
20 : Error

Explanation
The object authority manager has failed to locate the authority file <*insert_3*>. The migration of authority data cannot continue until the file has been restored. The queue manager will shutdown.

Response
Restore the authority file mentioned above and restart the queue manager.

AMQ5528
The IBM WebSphere MQ Object Authority Manager has failed to locate an object's authority file

Severity
20 : Error

Explanation
The object authority manager has failed to locate the authority file for the object <*insert_3*> of type (<*insert_1*>). The authority access to this object will initially be limited to members of the mqm group. Where type is one of the following:

1) Queue

2) Namelist

3) Process

5) Queue Manager

Response
To extend access to this object use the setmqaut command, see the IBM WebSphere MQ System Administration documentation for details.

AMQ5529
The Remote OAM Service is not available.

Severity
20 : Error

Explanation
The Remote OAM service is not available. The <*insert_1*> call returned <*insert_1*>, errno <*insert_2*> : <*insert_3*>. The context string is <*insert_4*>

Response
To extend access to this object use the setmqaut command, see the IBM WebSphere MQ System Administration documentation for details.

AMQ5600
Usage: crtmqm [-z] [-q] [-c Text] [-d DefXmitQ] [-h MaxHandles]

[-md DataPath] [-g ApplicationGroup]

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5600 (Tandem)
Usage: crtmqm [-z] [-q] [-c Text] [-d DefXmitQ] [-h MaxHandles]

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5600 (Windows)

Usage: crtmqm [-z] [-q] [-c Text] [-d DefXmitQ] [-h MaxHandles]

[-g ApplicationGroup]

[-ss | -sa | -si]

Severity

0 : Information

Explanation

This shows the correct usage.

Response

None.

AMQ5601

[-t TrigInt] [-u DeadQ] [-x MaxUMsgs] [-lp LogPri] [-ls LogSec]

Severity

0 : Information

Response

None.

AMQ5601 (Tandem)

[-t TrigInt] [-u DeadQ] [-x MaxUMsgs] [-m MIni] [-l CCSID]

Severity

0 : Information

Response

None.

AMQ5602

[-lc | -ll] [-lf LogFileSize] [-ld LogPath] QMgrName

Severity

0 : Information

Explanation

This shows the correct usage.

Response

None.

AMQ5602 (Tandem)

[-e NumECs] [-p QMVol] -n PMonProc -o HomeTerm

Severity

0 : Information

Response

None.

AMQ5602 (IBM i)

[-ll] [-lf LogFileSize] [-ld LogPath] [-lz ASPNum|ASPDev] QMgrName

Severity

0 : Information

Explanation

This shows the correct usage.

Response

None.

AMQ5603
Usage: dltmqm [-z] QMgrName

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5604
Usage: dspmqaut [-m QMgrName] [-n ObjName] -t ObjType (-p Principal | -g Group) [-s ServiceComponent]

Severity
0 : Information

Response
None.

AMQ5605
Usage: endmqm [-z] [-c | -w | -i | -p] [-s] QMgrName

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5605 (Tandem)
Usage: endmqm [-z] [-c | -i | -p] QMgrName

Severity
0 : Information

Response
None.

AMQ5606
Usage: setmqaut [-m QMgrName] [-n ObjName] -t ObjType (-p Principal | -g Group) [-s ServiceComponent] Authorizations

Severity
0 : Information

Response
None.

AMQ5607
Usage: strmqm [-a|-c|-p|-r] [-d none|minimal|all] [-z] [-ns] [QMgrName]

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5607 (Windows)

Usage: strmqm [-a | -c | -r | -p] [-d none|minimal|all] [-z]

[-ns] [-ss | -si] [QMgrName]

Severity

0 : Information

Explanation

This shows the correct usage.

Response

None.

AMQ5608

Usage: dspmqtrn [-m QMgrName] [-e] [-i] [-h]

Severity

0 : Information

Explanation

This shows the correct usage.

Response

None.

AMQ5609

Usage: rsvmqtrn -m QMgrName (-a | ((-b | -c | -f | -r RMId) Transaction,Number))

Severity

0 : Information

Explanation

This shows the correct usage.

Response

None.

AMQ5610 (Tandem)

Usage: strmqtrc [-m QMgrName] [-t TraceType]

Severity

0 : Information

Response

None.

AMQ5610 (Windows, UNIX and Linux)

Usage: strmqtrc [-m QMgrName] [-t TraceType] [-x TraceType] [-s] [-l MaxFileSize] [-e]

[-p ProgramName] [-i Pid.Tid] [-d UserDataSize] [-b StartTrigger] [-c StopTrigger]

Severity

0 : Information

Explanation

This applies to Windows, UNIX and Linux systems. MaxFileSize is the maximum size of a trace file in millions of bytes. UserDataSize is the size of user data to be traced in bytes.

Response

None.

AMQ5610 (IBM i)

Usage: strmqtrc [-m QMgrName] [-t TraceType] [-x TraceType] [-s] [-l MaxFileSize] [-e]

[-p ProgramName] [-i Pid.Tid] [-d UserDataSize] [-b StartTrigger] [-c StopTrigger]

[-o mqm | pex | all]

Severity
0 : Information

Explanation
None.

Response
None.

AMQ5611 (Tandem)
Usage: endmqtrc [-m QMgrName] [-a]

Severity
0 : Information

Response
None.

AMQ5611 (Windows)
Usage: endmqtrc [-p ProgramName] [-i Pid.Tid] [-m QMgrName] [-a] [-e]

Severity
0 : Information

Explanation
This applies to Windows, UNIX and Linux systems.

Response
None.

AMQ5611 (IBM i)
Usage: endmqtrc [-p ProgramName] [-i Pid.Tid] [-m QMgrName] [-a] [-e] [-o mqm | pex | all]

Severity
0 : Information

Explanation
This applies to AS/400 systems. MaxFileSize is the maximum size of a trace file in millions of bytes. UserDataSize is the size of user data to be traced in bytes.

Response
None.

AMQ5612
Usage: dspmqtrc [-t TemplateFile] [-hs] [-o OutputFileName] [-C InputFileCCSID] InputFileName(s)

Severity
0 : Information

Explanation
Options: -t Template file for formatting trace data -h Skip the trace file header -s Summary (format only the trace header) -o Save trace output to file -C Specifies the CCSID value for the input file

Response
None.

AMQ5613
Usage: dspmq [-m QMgrName] [-o status | -s] [-o default]

Severity
0 : Information

AMQ5614
Usage: setmqtry

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5615
Default objects cannot be created: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
During the creation of a queue manager, using the crtmqm command, the default objects could not be created. Possible reasons for this include another command, issued elsewhere, quiescing or stopping the queue manager, or insufficient storage being available.

Response
Use the Completion and Reason codes shown in the message to determine the cause of the failure, then re-try the command.

AMQ5616
Usage: setmqprd LicenseFile

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5617
Default objects cannot be created.

Severity
20 : Error

Explanation
During the creation of a queue manager using the crtmqm command, the default objects could not be created. The most likely reason for this error is that the queue manager was started before the crtmqm command had completed.

Response
Ensure that the queue manager being created is not started before the create request completes. Stop the queue manager if it is already running. Restart the queue manager using the strmqm command with the '-c' option to request that the default objects are created.

AMQ5618
integer

Severity
0 : Information

AMQ5619
string

Severity
0 : Information

AMQ5620
channel_name

Severity
0 : Information

AMQ5621
process_name

Severity
0 : Information

AMQ5622
q_name

Severity
0 : Information

AMQ5623
connection_name

Severity
0 : Information

AMQ5624
generic_channel_name

Severity
0 : Information

AMQ5625
generic_process_name

Severity
0 : Information

AMQ5626
generic_q_name

Severity
0 : Information

AMQ5627
qalias_name

Severity
0 : Information

AMQ5628
qmodel_name

Severity
0 : Information

AMQ5629
qlocal_name

Severity
0 : Information

AMQ5630
qremote_name

Severity
0 : Information

AMQ5631
namelist_name

Severity
0 : Information

AMQ5632
generic_namelist_name

Severity
0 : Information

AMQ5633
generic_Q_Mgr_name

Severity
0 : Information

AMQ5634
generic_cluster_name

Severity
0 : Information

AMQ5635
The argument supplied with the <*insert_3*> flag is not valid.

Severity
20 : Error

Explanation
The argument supplied with the -l parameter must be in the range 1 - 4293. The argument supplied with the -d parameter must be -1, 0 or greater than 15.

Response
Submit the command again with a valid argument.

AMQ5636
cluster_name

Severity
0 : Information

AMQ5638 (Tandem)
Usage: cleanrdf -b BkpSysName [-m QMgrName]

Severity
0 : Information

Response
None.

AMQ5639 (Tandem)
-s Status Server Proc -v Queue Server Proc QMgrName

Severity
0 : Information

Response
None.

AMQ5640 (Tandem)
Usage: altmqusr -m QMgrName -p Principal (-u UserName | -r)

Severity
0 : Information

Response
None.

AMQ5641 (Tandem)
Principal Userid Username Alias GroupName GroupType

Severity
0 : Information

AMQ5642 (Tandem)
The Principal name was specified incorrectly.

Severity
0 : Information

Explanation
The specified Principal name does not conform to the rules required by MQSeries.

Response
Correct the name and submit the command again.

AMQ5643 (Tandem)
Error modifying an entry in the Principal database.

Severity
0 : Information

Explanation
MQSeries was unable to update or delete the specified entry in the Principal database.

Response
Make sure that the entry for this Principal exists and submit the command again.

AMQ5644 (Tandem)
Usage: dspmqusr -m QMgrName [-p Principal]

Severity
0 : Information

Response
None.

AMQ5645 (Tandem)
The Tandem User name was specified incorrectly.

Severity
0 : Information

Explanation
The specified Tandem User name does not conform to the rules required by MQSeries.

Response
Correct the name and submit the command again.

AMQ5646
Usage: setmqcap Processors

Severity
0 : Information

AMQ5647
Usage: dspmqcap

Severity
0 : Information

AMQ5648
Usage: dmpmqaut [-m QMgrName] [-n Profile | -l] [-t ObjType] [-p Principal | -g Group] [-s ServiceComponent] [-e | -x]

Severity
0 : Information

Response
None.

AMQ5649
generic_authinfo_name

Severity
0 : Information

AMQ5650
authinfo_name

Severity
0 : Information

AMQ5651
qmname

Severity
0 : Information

AMQ5652
The Deferred Message process failed to connect to the WebSphere MQ queue manager for reason <*insert_1*>.

Severity
30 : Severe error

Explanation
The IBM WebSphere MQ queue manager <*insert_3*> might have generated earlier messages or FFST information explaining why the deferred message process (amqzdmaa) could not connect.

Response
Correct any configuration errors. Configuration errors that can cause this problem include badly configured CLWL Exit modules. If the problem persists save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5653
The mqm user is not defined.

Severity
30 : Severe error

Explanation
The system call getpwnam("mqm") failed with errno <*insert_1*>. The program was running as <*insert_3*>.

Response
Create the mqm user as a member of the mqm group and retry the operation.

AMQ5654
Usage: dspmqrte [-c] [-n] [-l Persistence] [-m QMgrName] [-o] [-p Priority]

Severity
0 : Information

Explanation
This shows the correct usage of the DSPMQRTE command.

Response
None.

AMQ5655
[-rq ReplyQName [-rqm ReplyQMgrName]] [-ro ReportOptions]

Severity
0 : Information

Explanation
This shows the correct usage of the DSPMQRTE command.

Response
None.

AMQ5656
[-xs Expiry] [-xp Pass] [-qm TargetQMgrName] [-ac [-ar]]

Severity
0 : Information

Explanation
This shows the correct usage of the DSPMQRTE command.

Response
None.

AMQ5657
[-d Delivery] [-f Forwarding] [-s Activities] [-t Detail]

Severity
0 : Information

Explanation
This shows the correct usage of the DSPMQRTE command.

Response
None.

AMQ5658
[-i CorrelId] [-b] [-v Verbosity] [-w WaitTime]

Severity
0 : Information

Explanation
This shows the correct usage of the DSPMQRTE command.

Response
None.

AMQ5659 (UNIX and Linux)
Unable to access trace shared memory: <*insert_1*>

Severity
0 : Information

Explanation
This applies to UNIX and Linux systems.

Response
Refer to IBM Service Personnel

AMQ5659 (IBM i)
Unable to access trace control shared memory (<*insert_1*>)

Severity
0 : Information

Explanation
An unexpected error accessing trace control memory has occurred whilst attempting to start or stop trace. The attempt to access trace control failed with a return code of <*insert_1*>.

Response
Contact your IBM representative.

AMQ5660
-q TargetQName | -ts TargetTopicString

Severity
0 : Information

Explanation
This shows the correct usage of the DSPMQRTE command.

Response
None.

AMQ5675
Inconsistent use of installations detected.

Severity
20 : Error

Explanation
When executing program <*insert_3*> from installation <*insert_4*>, IBM WebSphere MQ detected that due to the configuration of the environment resources were loaded from installation <*insert_5*>. The program cannot complete successfully while the program is executing using inconsistent installations.

Response
If applicable, run program <*insert_3*> from installation <*insert_5*> or configure the environment so that all resources required by program <*insert_3*> are loaded from installation <*insert_4*>.

AMQ5688
Unable to associate queue manager <*insert_3*> with installation <*insert_4*>.

Severity
20 : Error

Explanation
The request to associate queue manager <*insert_3*> with installation <*insert_4*> failed. This could be caused by the MQ version with which the queue manager was previously running being greater than the version of installation <*insert_4*>.

Response
Check that the installation specified is as intended and reissue the command.

AMQ5691
Queue manager <*insert_4*> is associated with a different installation.

Severity
20 : Error

Explanation

The command <*insert_3*> was issued against queue manager <*insert_4*>, but the queue manager is associated with a different installation than the one currently in use, <*insert_5*>. In order for the command to succeed, the installation that the command is executing from must match the installation that the queue manager is associated with.

Response

Either change the installation the command is being executed from using the setmqenv command or associate the queue manager with the current installation using the setmqm command.

AMQ5700

listener_name

Severity

0 : Information

AMQ5701

service_name

Severity

0 : Information

AMQ5749

display_cmd

Severity

0 : Information

AMQ5750

filter_keyword

Severity

0 : Information

AMQ5751

operator

Severity

0 : Information

AMQ5752

filter_value

Severity

0 : Information

AMQ5753

topic_name

Severity

0 : Information

AMQ5754

obj_name

Severity

0 : Information

AMQ5755

generic_topic_name

Severity

0 : Information

AMQ5756

subscription_name

Severity
0 : Information

AMQ5757
subscription_id

Severity
0 : Information

AMQ5758
generic_topic_string

Severity
0 : Information

AMQ5765
channel_profile

Severity
0 : Information

AMQ5805
IBM WebSphere MQ Publish/Subscribe broker currently running for queue manager.

Severity
10 : Warning

Explanation
The command was unsuccessful because queue manager <*insert_3*> currently has an IBM WebSphere MQ Publish/Subscribe broker running.

Response
None.

AMQ5806
IBM WebSphere MQ Publish/Subscribe broker started for queue manager <*insert_3*>.

Severity
0 : Information

Explanation
IBM WebSphere MQ Publish/Subscribe broker started for queue manager <*insert_3*>.

Response
None.

AMQ5807
IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> ended.

Severity
0 : Information

Explanation
The IBM WebSphere MQ Publish/Subscribe broker on queue manager <*insert_3*> has ended.

Response
None.

AMQ5808
IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> is already quiescing.

Severity
10 : Warning

Explanation
The endmqbrk command was unsuccessful because an orderly shutdown of the IBM WebSphere MQ Publish/Subscribe broker running on queue manager <*insert_3*> is already in progress.

Response
None.

AMQ5808 (IBM i)
IBM WebSphere MQ Publish/Subscribe broker is already quiescing.

Severity
10 : Warning

Explanation
The endmqbrk command was unsuccessful because an orderly shutdown of the broker, running on queue manager <*insert_3*>, is already in progress.

Response
None.

AMQ5809
IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> starting.

Severity
0 : Information

Explanation
The dspmqbrk command has been issued to query the state of the IBM WebSphere MQ Publish/Subscribe broker. The IBM WebSphere MQ Publish/Subscribe broker is currently initializing.

Response
None.

AMQ5810
IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> running.

Severity
0 : Information

Explanation
The dspmqbrk command has been issued to query the state of the IBM WebSphere MQ Publish/Subscribe broker. The IBM WebSphere MQ Publish/Subscribe broker is currently running.

Response
None.

AMQ5811
IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> quiescing.

Severity
0 : Information

Explanation
The dspmqbrk command has been issued to query the state of the IBM WebSphere MQ Publish/Subscribe broker. The IBM WebSphere MQ Publish/Subscribe broker is currently performing a controlled shutdown.

Response
None.

AMQ5812
IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> stopping.

Severity
0 : Information

Explanation
Either the dspmqbrk command or the endmqbrk command has been issued. The IBM WebSphere MQ Publish/Subscribe broker is currently performing an immediate shutdown. If the endmqbrk command has been issued to request that the broker terminate, the command is unsuccessful because the broker is already performing an immediate shutdown.

Response
None.

AMQ5813
IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> not active.

Severity
0 : Information

Explanation
An IBM WebSphere MQ Publish/Subscribe broker administration command has been issued to query or change the state of the broker. The WebSphere MQ Publish/Subscribe broker is not currently running.

Response
None.

AMQ5814
IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> ended abnormally.

Severity
0 : Information

Explanation
The dspmqbrk command has been issued to query the state of the IBM WebSphere MQ Publish/Subscribe broker. The IBM WebSphere MQ Publish/Subscribe broker has ended abnormally.

Response
Refer to the queue manager error logs to determine why the broker ended abnormally.

AMQ5815
Invalid IBM WebSphere MQ Publish/Subscribe broker initialization file stanza for queue manager (<*insert_3*>).

Severity
20 : Error

Explanation
The broker was started using the strmqbrk command. The broker stanza in the queue manager initialization file is not valid. The broker will terminate immediately. The invalid attribute is <*insert_5*>.

Response
Correct the broker stanza in the queue manager initialization file.

AMQ5815 (Windows)
The IBM WebSphere MQ Publish/Subscribe broker configuration for queue manager (<*insert_3*>) is not valid.

Severity
20 : Error

Explanation
The broker was started using the strmqbrk command. The broker configuration information is not valid. The broker will terminate immediately. The invalid attribute is <*insert_5*>.

Response
Correct the broker attribute using the cfgmqbrk configuration tool.

AMQ5815 (IBM i)
Invalid IBM WebSphere MQ Publish/Subscribe broker initialization file stanza.

Severity
20 : Error

Explanation
The broker was started using the strmqbrk command. The Broker stanza in the queue manager(<*insert_3*>) initialization file is not valid. The broker will terminate immediately. The invalid attribute is <*insert_5*>.

Response
Correct the Broker stanza in the queue manager initialization file.

AMQ5816
Unable to open IBM WebSphere MQ Publish/Subscribe broker control queue for reason <*insert_1*>,<*insert_2*>.

Severity
20 : Error

Explanation
The broker has failed to open the broker control queue (<*insert_3*>). The attempt to open the queue failed with completion code <*insert_1*> and reason <*insert_2*>. The most likely reasons for this error are that an application program has opened the broker control queue for exclusive access, or that the broker control queue has been defined incorrectly. The broker will terminate immediately.

Response
Correct the problem and restart the broker.

AMQ5817
An invalid stream queue has been detected by the broker.

Severity
10 : Warning

Explanation
IBM WebSphere MQ has detected an attempt to use a queue (<*insert_3*>) as a stream queue, but the attributes of the queue make it unsuitable for use as a stream queue. The most likely reason for this error is that the queue is: (1) Not a local queue; (2) A shareable queue; (3) A temporary dynamic queue. If the queue was created using implicit stream creation, the model stream might have been defined incorrectly. The message that caused the stream to be created will be rejected or put to the dead-letter queue, depending upon the message report options and broker configuration.

Response
Correct the problem and resubmit the request.

AMQ5818
Unable to open IBM WebSphere MQ Publish/Subscribe broker stream queue.

Severity
10 : Warning

Explanation
The broker has failed to open a stream queue (<*insert_3*>). The attempt to open the queue failed

with completion code <*insert_1*> and reason <*insert_2*>. The most likely reasons for this error are (1) a new stream name has been added to SYSTEM.QPUBSUB.QUEUE.NAMELIST but the stream queue does not exist (2) an application has the queue open for exclusive access.

Response
Correct the problem.

AMQ5819
An IBM WebSphere MQ Publish/Subscribe broker stream has ended abnormally.

Severity
10 : Warning

Explanation
The broker stream (<*insert_3*>) has ended abnormally for reason <*insert_1*>. The broker will attempt to restart the stream. If the stream should repeatedly fail then the broker will progressively increase the time between attempts to restart the stream.

Response
Investigate why the problem occurred and take appropriate action to correct the problem. If the problem persists, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5820
IBM WebSphere MQ Publish/Subscribe broker stream (<*insert_3*>) restarted.

Severity
0 : Information

Explanation
The broker has restarted a stream that ended abnormally. This message will frequently be preceded by message AMQ5867 or AMQ5819 indicating why the stream ended.

Response
Correct the problem.

AMQ5821
IBM WebSphere MQ Publish/Subscribe broker unable to contact parent broker.

Severity
10 : Warning

Explanation
The broker has been started specifying a parent broker. The broker has been unable to send a message to the parent broker (<*insert_3*>) for reason <*insert_1*>.

Response
Investigate why the problem occurred and take appropriate action to correct the problem. The problem is likely to be caused by the parent broker name not resolving to the name of a transmission queue on the local broker.

AMQ5822
IBM WebSphere MQ Publish/Subscribe broker failed to register with parent broker.

Severity
10 : Warning

Explanation
The broker has been started specifying a parent broker (<*insert_3*>). The broker attempted to register as a child of the parent broker, but received an exception response (<*insert_1*>) indicating that this was not possible. The broker will attempt to reregister as a child of the parent

periodically. The child might not be able to process global publications or subscriptions correctly until this registration process has completed normally.

Response

Investigate why the problem occurred and take appropriate action to correct the problem. The problem is likely to be caused by the parent broker not yet existing, or a problem with the SYSTEM.BROKER.INTER.BROKER.COMMUNICATIONS queue at the parent broker.

AMQ5823

Exit path attribute invalid in IBM WebSphere MQ Publish/Subscribe broker stanza.

Severity

10 : Warning

Explanation

The broker exit path attribute <*insert_3*> is not valid. The attribute should be specified as: <path><module name>(<function name>). The broker will terminate immediately.

Response

Correct the problem with the attribute and restart the broker.

AMQ5825

The address of the IBM WebSphere MQ Publish/Subscribe broker exit function could not be found.

Severity

10 : Warning

Explanation

The address of the broker exit function <*insert_4*> could not be found in module <*insert_3*> for reason <*insert_1*>:<*insert_5*>. The broker will terminate immediately.

Response

Correct the problem with the broker exit function <*insert_4*> in module <*insert_3*>, and restart the broker.

AMQ5826

IBM WebSphere MQ Publish/Subscribe has failed to propagate a subscription to another queue manager.

Severity

10 : Warning

Explanation

The queue manager failed to propagate subscription to stream (<*insert_4*>) at broker (<*insert_3*>). Reason codes <*insert_1*> and <*insert_2*>. An application has either registered or deregistered a subscription to stream (<*insert_4*>). The queue manager has attempted to propagate the subscription change to broker (<*insert_3*>) but the request has not been successful. Messages published on stream (<*insert_4*>) through queue manager (<*insert_3*>) might not reach this queue manager.

Response

Use the reason codes to investigate why the problem occurred and take appropriate action to correct the problem. Use the command REFRESH QMGR TYPE(PROXYSUB) to refresh proxy subscriptions. ????????

AMQ5827

An IBM WebSphere MQ Publish/Subscribe broker internal subscription has failed.

Severity

10 : Warning

Explanation

The broker failed to subscribe to stream (<*insert_4*>) at broker (<*insert_3*>) with reason codes

<*insert_1*> and <*insert_2*>. Related brokers learn about each others configuration by subscribing to information published by each other. A broker has discovered that one of these internal subscriptions has failed. The broker will reissue the subscription immediately. The broker cannot function correctly without knowing some information about neighboring brokers. The information that this broker has about broker (<*insert_3*>) is not complete and this could lead to subscriptions and publications not being propagated around the network correctly.

Response

Investigate why the problem occurred and take appropriate action to correct the problem. The most likely cause of this failure is a problem with the SYSTEM.BROKER.CONTROL.QUEUE at broker (<*insert_3*>), or a problem with the definition of the route between this broker and broker (<*insert_3*>).

AMQ5828

IBM WebSphere MQ Publish/Subscribe broker exit returned an ExitResponse that is not valid.

Severity

10 : Warning

Explanation

The broker exit returned an ExitResponse <*insert_1*> that is not valid. The message has been allowed to continue and an FFST has been generated that contains the entire exit parameter structure.

Response

Correct the problem with the broker exit.

AMQ5829

Usage: amqfqpub [-m QMgrName]. Do not run this command manually.

Severity

0 : Information

Explanation

This shows the correct usage.

Response

None.

AMQ5830

The endmqbrk command can no longer be used. The &MQQPUBSUB_short is enabled/disabled by altering the Queue manager's PSMODE attribute. Setting PSMODE to "COMPAT" disables the queued pubsub interface.

Severity

0 : Information

Explanation

The endmqbrk command (shipped with earlier versions of MQ) is no longer used to enable/disable the IBM WebSphere MQ Publish/Subscribe Instead of issuing the endmqbrk command the PSMODE attribute of the queue manager should be set to COMPAT.

Response

None.

AMQ5832

IBM WebSphere MQ Publish/Subscribe broker failed to publish configuration information on SYSTEM.BROKER.ADMIN.STREAM.

Severity

10 : Warning

Explanation

Related brokers learn about each others configuration by subscribing to information published by

each other. A broker has discovered that one of these internal publications has failed. The broker will republish the information immediately. Brokers cannot function correctly without knowing some information about neighboring brokers. The information that neighboring brokers have of this broker might not be complete and this could lead to some subscriptions and publications not being propagated around the network.

Response
Investigate why the problem occurred and take appropriate action to correct the problem.

AMQ5833
A loop has been detected in the IBM WebSphere MQ Publish/Subscribe broker hierarchy.

Severity
20 : Error

Explanation
The broker, on queue manager (<*insert_3*>), introduced a loop in the broker hierarchy. This broker will terminate immediately.

Response
Remove broker (<*insert_3*>) from the hierarchy, either by deleting the broker, or by removing knowledge of the broker's parent, using the clrmqbrk command.

AMQ5834
Conflicting queue manager names in the IBM WebSphere MQ Publish/Subscribe broker hierarchy.

Severity
10 : Warning

Explanation
The names of the queue managers (<*insert_3*>) and (<*insert_4*>) in the broker hierarchy both start with the same 12 characters. The first 12 characters of a broker's queue manager name should be unique to ensure that no confusion arises within the broker hierarchy, and to guarantee unique message ID allocation.

Response
Use a queue manager naming convention that guarantees uniqueness of the first 12 characters of the queue manager name.

AMQ5835
IBM WebSphere MQ Publish/Subscribe broker failed to inform its parent of a relation for reason <*insert_1*>.

Severity
0 : Information

Explanation
The failed to notify its parent on queue manager (<*insert_3*>) of the relation (<*insert_4*>) in the broker hierarchy. The notification message will be put to the parent's dead-letter queue. A failure to notify a broker of a new relation will mean that no loop detection can be performed for the new relation.

Response
Diagnose and correct the problem on the parent queue manager. One possible reason for this is that the parent broker does not yet exist.

AMQ5836
Duplicate queue manager name located in the IBM WebSphere MQ Publish/Subscribe hierarchy.

Severity
0 : Information

Explanation

Multiple instances of the queue manager name (<*insert_3*>) have been located. This could either be the result of a previously resolved loop in the broker hierarchy, or multiple queue managers in the broker hierarchy having the same name.

Response

If this broker introduced a loop in the hierarchy (typically identified by message AMQ5833), this message can be ignored. It is strongly recommended that every queue manager in a broker hierarchy has a unique name. It is not recommended that multiple queue managers use the same name.

AMQ5837

IBM WebSphere MQ Publish/Subscribe broker failed to quiesce queue (<*insert_3*>) for reason <*insert_1*>.

Severity

10 : Warning

Explanation

When a broker is deleted, the broker's input queues are quiesced by making the queue get inhibited, and writing the contents of the queue to the dead-letter queue (depending upon the report options of the message). The broker was unable to quiesce the named queue for the reason shown. The attempt to delete the broker will fail.

Response

Investigate why the problem occurred, take appropriate action to correct the problem, and reissue the dltmqbrk command. Likely reasons include the queue being open for input by another process, there being no dead-letter queue defined at this queue manager, or the operator setting the queue to get inhibited while the dltmqbrk command is running. If there is no dead-letter queue defined, the reason will be reported as MQRC_UNKNOWN_OBJECT_NAME. If the problem occurs because there is no dead-letter queue defined at this broker, the operator can either define a dead-letter queue, or manually empty the queue causing the problem.

AMQ5837 (IBM i)

IBM WebSphere MQ Publish/Subscribe broker failed to quiesce queue.

Severity

10 : Warning

Explanation

When a broker is deleted, the broker's input queues are quiesced by making the queue get inhibited, and writing the contents of the queue to the dead-letter queue (depending upon the report options of the message). The broker was unable to quiesce the queue (<*insert_3*>) for reason <*insert_1*>. The attempt to delete the broker will fail.

Response

Investigate why the problem occurred, take appropriate action to correct the problem, and reissue the dltmqbrk command. Likely reasons include the queue being open for input by another process, there being no dead-letter queue defined at this queue manager, or the operator setting the queue to get inhibited while the dltmqbrk command is running. If there is no dead-letter queue defined, the reason will be reported as MQRC_UNKNOWN_OBJECT_NAME. If the problem occurs because there is no dead-letter queue defined at this broker, the operator can either define a dead-letter queue, or manually empty the queue causing the problem.

AMQ5838

IBM WebSphere MQ Publish/Subscribe broker cannot be deleted.

Severity

10 : Warning

Explanation

The broker cannot be deleted as child (<*insert_3*>) is still registered. A broker cannot be deleted until all other brokers that have registered as children of that broker, have deregistered as its children.

Response

Use the clrmqbrk and dltmqbrk commands to change the broker topology so that broker (<*insert_3*>) is not registered as a child of the broker being deleted.

AMQ5839

IBM WebSphere MQ Publish/Subscribe broker received an unexpected inter-broker communication.

Severity

10 : Warning

Explanation

A broker has received an inter-broker communication that it did not expect. The message was sent by broker (<*insert_3*>). The message will be processed according to the report options in that message. The most likely reason for this message is that the broker topology has been changed while inter-broker communication messages were in transit (for example, on a transmission queue) and that a message relating to the previous broker topology has arrived at a broker in the new topology. This message may be accompanied by an informational FFST including details of the unexpected communication.

Response

If the broker topology has changed and the broker named in the message is no longer related to the broker issuing this message, this message can be ignored. If the clrmqbrk command was issued to unilaterally remove knowledge of broker (<*insert_3*>) from this broker, the clrmqbrk command should also be used to remove knowledge of this broker from broker (<*insert_3*>). If the clrmqbrk command was issued to unilaterally remove knowledge of this broker from broker (<*insert_3*>), the clrmqbrk command should also be used to remove knowledge of broker (<*insert_3*>) at this broker.

AMQ5840

IBM WebSphere MQ Publish/Subscribe broker unable to delete queue.

Severity

10 : Warning

Explanation

The broker has failed to delete the queue (<*insert_3*>) for reason <*insert_2*>. The broker typically attempts to delete queues during dltmqbrk processing, in which case the dltmqbrk command will fail.

Response

The most likely reason for this error is that some other process has the queue open. Determine why the queue cannot be deleted, remove the inhibitor, and retry the failed operation. In a multi-broker environment, it is likely that a message channel agent might have queues open, which the broker needs to delete for a dltmqbrk command to complete.

AMQ5841

IBM WebSphere MQ Publish/Subscribe broker (<*insert_3*>) deleted.

Severity

0 : Information

Explanation

The broker (<*insert_3*>) has been deleted using the dltmqbrk command.

Response

None.

AMQ5842
IBM WebSphere MQ Publish/Subscribe broker (<*insert_3*>) cannot be deleted for reason <*insert_1*>:<*insert_5*>.

Severity
20 : Error

Explanation
An attempt has been made to delete the broker (<*insert_3*>) but the request has failed for reason <*insert_1*>:<*insert_5*>.

Response
Determine why the dltmqbrk command cannot complete successfully. The message logs for the queue manager might contain more detailed information on why the broker cannot be deleted. Resolve the problem that is preventing the command from completing and reissue the dltmqbrk command.

AMQ5842 (IBM i)
IBM WebSphere MQ Publish/Subscribe broker cannot be deleted.

Severity
20 : Error

Explanation
An attempt has been made to delete the IBM WebSphere MQ Publish/Subscribe broker (<*insert_3*>) but the request has failed for reason <*insert_1*>:<*insert_5*>.

Response
Determine why the dltmqbrk command cannot complete successfully. The message logs for the queue manager might contain more detailed information on why the broker cannot be deleted. Resolve the problem that is preventing the command from completing and reissue the dltmqbrk command.

AMQ5843
IBM WebSphere MQ Publish/Subscribe broker (<*insert_3*>) cannot be started as it is partially deleted.

Severity
10 : Warning

Explanation
An attempt has been made to start a broker that is in a partially deleted state. An earlier attempt to delete the broker has failed. The broker deletion must be completed before the broker will be allowed to restart. When broker deletion is successful, message AMQ5841 is issued, indicating that the broker has been deleted. If this message is not received on completion of a dltmqbrk command, the broker deletion has not been completed and the command will have to be reissued.

Response
Investigate why the earlier attempt to delete the broker failed. Resolve the problem and reissue the dltmqbrk command.

AMQ5843 (IBM i)
IBM WebSphere MQ Publish/Subscribe broker cannot be started as it is partially deleted.

Severity
10 : Warning

Explanation
An attempt has been made to start the broker <*insert_3*> that is in a partially deleted state. An earlier attempt to delete the broker has failed. The broker deletion must be completed before the broker will be allowed to restart. When broker deletion is successful, message AMQ5841 is

issued, indicating that the broker has been deleted. If this message is not received on completion of a dltmqbrk command, the broker deletion has not been completed and the command will have to be reissued.

Response
Investigate why the earlier attempt to delete the broker failed. Resolve the problem and reissue the dltmqbrk command.

AMQ5844
The relation between two IBM WebSphere MQ Publish/Subscribe brokers is unknown.

Severity
10 : Warning

Explanation
The clrmqbrk command has been issued in an attempt to remove a brokers knowledge of a relation of that broker. The relative (<*insert_4*>) is unknown at broker (<*insert_3*>). If the "-p" flag was specified, the broker does not currently have a parent. If the "-c" flag was specified, the broker does not recognize the named child.

Response
Investigate why the broker is unknown.

AMQ5845
Usage: dltmqbrk -m QMgrName

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5847
IBM WebSphere MQ Publish/Subscribe broker (<*insert_3*>) has removed knowledge of relation (<*insert_4*>).

Severity
0 : Information

Explanation
The clrmqbrk command has been used to remove knowledge of broker (<*insert_4*>) from broker (<*insert_3*>).

Response
None.

AMQ5847 (IBM i)
IBM WebSphere MQ Publish/Subscribe broker relation removed.

Severity
0 : Information

Explanation
The clrmqbrk command has been used to remove knowledge of broker (<*insert_4*>) from broker (<*insert_3*>).

Response
None.

AMQ5848
IBM WebSphere MQ Publish/Subscribe broker (<*insert_3*>) has failed to remove references to relation (<*insert_4*>) for reason <*insert_1*>:<*insert_5*>.

Severity
20 : Error

Explanation
An attempt has been made to remove references to broker (<*insert_4*>) from broker (<*insert_3*>) using the clrmqbrk command, but the request has been unsuccessful.

Response
Determine why the clrmqbrk command cannot complete successfully. The message logs for the queue manager might contain more detailed information on why the broker cannot be deleted. Resolve the problem that is preventing the command from completing and then reissue the clrmqbrk command.

AMQ5848 (IBM i)
IBM WebSphere MQ Publish/Subscribe broker has failed to remove references to a related broker.

Severity
20 : Error

Explanation
An attempt has been made to remove references to broker (<*insert_4*>) from broker (<*insert_3*>) using the clrmqbrk command, but the request has been unsuccessful for reason <*insert_1*>:<*insert_5*>.

Response
Determine why the clrmqbrk command cannot complete successfully. The message logs for the queue manager might contain more detailed information on why the broker cannot be deleted. Resolve the problem that is preventing the command from completing and then reissue the clrmqbrk command.

AMQ5849
IBM WebSphere MQ Publish/Subscribe broker may not change parent.

Severity
10 : Warning

Explanation
An attempt has been made to start broker (<*insert_3*>), nominating broker (<*insert_4*>) as its parent. The broker (<*insert_3*>) has previously been started, nominating broker (<*insert_5*>) as its parent. The strmqbrk command cannot be used to change an existing relationship.

Response
Do not attempt to change the broker topology by using the strmqbrk command. The dltmqbrk and clrmqbrk commands are the only supported means of changing the broker topology. Refer to the documentation of those commands for guidance on changing the broker topology.

AMQ5850
IBM WebSphere MQ Publish/Subscribe broker interrupted while creating queue.

Severity
10 : Warning

Explanation
The broker was interrupted while creating queue (<*insert_3*>) for user ID (<*insert_4*>). When the broker creates a queue, it first creates the queue with default security attributes and it then sets the appropriate security attributes for the queue. If the broker should be interrupted during this operation (for example the queue manager is shut down), the broker cannot reliably detect that the security attributes have not been set correctly. The broker was creating a queue, but was interrupted before it could complete creation of the queue and setting the initial authority. If the interrupt occurred before the initial authority of the queue could be set, it might be necessary for the operator to set the appropriate authorities using the setmqaut command.

Response

Confirm that the named queue has the appropriate security attributes and modify them as necessary.

AMQ5851

IBM WebSphere MQ Publish/Subscribe broker interrupted while creating internal queue.

Severity

10 : Warning

Explanation

The broker was interrupted while creating internal queue (<*insert_3*>) for user ID (<*insert_4*>). When the broker creates an internal queue, it first creates the queue with default security attributes and it then sets the appropriate security attributes for the queue. If the broker should be interrupted during this operation (for example the queue manager is shut down), the broker attempts to delete and redefine the queue. If the internal queue is available to users (for example, the default stream or the administration stream), it is possible that a user will put a message on the queue while it is in this invalid state, or that a user application has the queue open. In this situation the broker does not automatically redefine the queue and cannot be restarted until the queue has been emptied or closed.

Response

Examine any messages on the named queue and take appropriate action to remove them from the queue. Ensure that no applications have the queue open.

AMQ5852

IBM WebSphere MQ Publish/Subscribe broker failed to propagate delete publication command.

Severity

0 : Information

Explanation

The broker failed to propagate delete publication command for stream (<*insert_3*>) to related broker (<*insert_4*>) for reason <*insert_1*>. When an application issues a delete publication command to delete a global publication, the command has to be propagated to all brokers in the sub-hierarchy supporting the stream. The broker reporting the error has failed to forward a delete publication command to a related broker (<*insert_4*>) who supports stream (<*insert_3*>). Delete publication commands are propagated without MQRO_DISCARD_MSG and the command message might have been written to a dead-letter queue. The topic for which the delete publication has failed is (<*insert_5*>).

Response

If the delete publication has failed because the stream has been deleted at the related broker, this message can be ignored. Investigate why the delete publication has failed and take the appropriate action to recover the failed command.

AMQ5853

IBM WebSphere MQ Publish/Subscribe failed to propagate a delete publication command.

Severity

0 : Information

Explanation

The broker failed to propagate a delete publication command for stream (<*insert_3*>) to a previously related broker. When an application issues a delete publication command to delete a global publication, the command is propagated to all brokers in the sub-hierarchy supporting the stream. The broker topology was changed after deleting the publication, but before a broker removed by the topology change processed the propagated delete publication message. The topic for which the delete publication has failed is (<*insert_5*>).

Response

It is the user's responsibility to quiesce broker activity before changing the broker topology using

the clrmqbrk command. Investigate why this delete publication activity was not quiesced. The delete publication command will have been written to the dead-letter queue at the broker that was removed from the topology. In this case, further action might be necessary to propagate the delete publication command that was not quiesced before the clrmqbrk command was issued. If this message occurs as a result of the dltmqbrk command, the publication will have been deleted as a result of the dltmqbrk command, and the delete publication message will have been written to the dead-letter queue at the queue manager where the broker was deleted. In this case the delete publication message on the dead-letter queue can be discarded.

AMQ5854

IBM WebSphere MQ Publish/Subscribe broker failed to propagate a delete publication command.

Severity

0 : Information

Explanation

When an application issues a delete publication command to delete a global publication, the command has to be propagated to all brokers in the sub-hierarchy supporting the stream. At the time the delete publication was propagated, broker (<*insert_4*>) was a known relation of this message broker supporting stream (<*insert_3*>). Before the delete publication command arrived at the related broker, the broker topology was changed so that broker (<*insert_4*>) no longer supported stream (<*insert_3*>). The topic for which the delete publication has failed is (<*insert_5*>).

Response

It is the user's responsibility to quiesce broker activity before changing the stream topology of the broker. Investigate why this delete publication activity was not quiesced. The delete publication command will have been written to the dead-letter queue at broker (<*insert_4*>).

AMQ5855

IBM WebSphere MQ Publish/Subscribe broker ended.

Severity

10 : Warning

Explanation

An attempt has been made to run the broker (<*insert_3*>) but the broker has ended for reason <*insert_1*>:<*insert_5*>.

Response

Determine why the broker ended. The message logs for the queue manager might contain more detailed information on why the broker cannot be started. Resolve the problem that is preventing the command from completing and reissue the strmqbrk command.

AMQ5856

Broker publish command message cannot be processed. Reason code <*insert_1*>.

Severity

10 : Warning

Explanation

The IBM WebSphere MQ Publish/Subscribe broker failed to process a publish message for stream (<*insert_3*>). The broker was unable to write the publication to the dead-letter queue and was not permitted to discard the publication. The broker will temporarily stop the stream and will restart the stream and consequently retry the publication after a short interval.

Response

Investigate why the error has occurred and why the publication cannot be written to the dead-letter queue. Either manually remove the publication from the stream queue, or correct the problem that is preventing the broker from writing the publication to the dead-letter queue.

AMQ5857

Broker control command message cannot be processed. Reason code <*insert_1*>.

Severity

10 : Warning

Explanation

The IBM WebSphere MQ Publish/Subscribe broker failed to process a command message on the SYSTEM.BROKER.CONTROL.QUEUE. The broker was unable to write the command message to the dead-letter queue and was not permitted to discard the command message. The broker will temporarily stop the stream and will restart the stream and consequently retry the command message after a short interval. Other broker control commands cannot be processed until this command message has been processed successfully or removed from the control queue.

Response

Investigate why the error has occurred and why the command message cannot be written to the dead-letter queue. Either, manually remove the command message from the stream queue, or correct the problem that is preventing the broker from writing the command message to the dead-letter queue.

AMQ5858

Broker could not send publication to subscriber queue.

Severity

10 : Warning

Explanation

A failure has occurred sending a publication to subscriber queue (<*insert_4*>) at queue manager (<*insert_3*>) for reason <*insert_1*>. The broker configuration options prevent it from recovering from this failure by discarding the publication or by sending it to the dead-letter queue. Instead the broker will back out the unit of work under which the publication is being sent and retry the failing command message a fixed number of times. If the problem still persists, the broker will then attempt to recover by failing the command message with a negative reply message. If the issuer of the command did not request negative replies, the broker will either discard or send to the dead-letter queue the failing command message. If the broker configuration options prevent this, the broker will restart the affected stream, which will reprocess the failing command message again. This behavior will be repeated until such time as the failure is resolved. During this time the stream will be unable to process further publications or subscriptions.

Response

Usually the failure will be due to a transient resource problem, for example, the subscriber queue, or an intermediate transmission queue, becoming full. Use reason code <*insert_1*> to determine what remedial action is required. If the problem persists for a long time, you will notice the stream being continually restarted by the broker. Evidence of this occurring will be a large number of AMQ5820 messages, indicating stream restart, being written to the error logs. In such circumstances, manual intervention will be required to allow the broker to dispose of the failing publication. To do this, you will need to end the broker using the endmqbrk command and restart it with appropriate disposition options. This will allow the publication to be sent to the rest of the subscribers, while allowing the broker to discard or send to the dead-letter queue the publication that could not be sent.

AMQ5859

IBM WebSphere MQ Publish/Subscribe broker stream is terminating due to an internal resource problem.

Severity

10 : Warning

Explanation

The broker stream (<*insert_3*>) has run out of internal resources and will terminate with reason code <*insert_1*>. If the command in progress was being processed under sync point control, it will

be backed out and retried when the stream is restarted by the broker. If the command was being processed out of sync point control, it will not be able to be retried when the stream is restarted.

Response

This message should only be issued in very unusual circumstances. If this message is issued repeatedly for the same stream, and the stream is not especially large in terms of subscriptions, topics, and retained publications, save all generated diagnostic information and use either the IBM WebSphere MQ support web page, or the IBM support assistant at the IBM SupportAssistant web page, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5862

IBM WebSphere MQ Publish/Subscribe broker for queue manager <*insert_3*> migrating.

Severity

0 : Information

Explanation

The dspmqbrk command has been issued to query the state of the broker. The broker is currently being migrated.

Response

None.

AMQ5863

WebSphere Brokers broker not ready for migration. See message logs for guidance.

Severity

10 : Warning

Explanation

The migmqbrk command was unsuccessful because the WebSphere Brokers broker was not ready to accept messages. The state of the WebSphere MQ Publish/Subscribe message broker is exported to the WebSphere Brokers broker in a series of messages sent to queue SYSTEM.BROKER.INTERBROKER.QUEUE. Before migration commences the IBM WebSphere MQ Publish/Subscribe broker checks whether the WebSphere Brokers broker is ready to accept messages on this queue. This check has failed for reason <*insert_1*> so migration has been abandoned.

Response

Reason code <*insert_1*> should be used to determine the nature of the problem. A value of 1 means that queue SYSTEM.BROKER.INTERBROKER.QUEUE does not exist. This is probably because no WebSphere Brokers broker has been defined yet on this queue manager. A value of 2 means that the WebSphere Brokers broker does not have the queue open probably because it hasn't been started or the first message flow has yet to be deployed for it. If both of these steps have been taken then the WebSphere Brokers broker may have been created incorrectly. In particular, it should have been created in migration mode. If the broker was not created with the migration flag set then it will need to be deleted and re-created before migration can commence. For any other value in the reason code, use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Note that until the problem has been resolved the IBM WebSphere MQ Publish/Subscribe broker can still be restarted with the the strmqbrk command.

AMQ5864

Broker reply message could not be sent. The command will be retried.

Severity

10 : Warning

Explanation

While processing a publish/subscribe command, the IBM WebSphere MQ Publish/Subscribe broker could not send a reply message to queue (<*insert_4*>) at queue manager (<*insert_3*>) for reason <*insert_1*>. The broker was also unable to write the message to the dead-letter queue. Since the command is being processed under sync point control, the broker will attempt to retry the command in the hope that the problem is only of a transient nature. If, after a set number of retries, the reply message still could not be sent, the command message will be discarded if the report options allow it. If the command message is not discardable, the stream will be restarted, and processing of the command message recommenced.

Response

Use reason code <*insert_1*> to determine what remedial action is required. If the failure is due to a resource problem (for example, a queue being full), you might find that the problem has already cleared itself. If not, this message will be issued repeatedly each time the command is retried. In this case you are strongly advised to define a dead-letter queue to receive the reply message so that the broker can process other commands while the problem is being investigated. Check the application from which the command originated and ensure that it is specifying its reply-to queue correctly.

AMQ5865

Broker reply message could not be sent.

Severity

10 : Warning

Explanation

While processing a publish/subscribe command, the IBM WebSphere MQ Publish/Subscribe broker could not send a reply message to queue (<*insert_4*>) at queue manager (<*insert_3*>) for reason <*insert_1*>. The broker was also unable to write the message to the dead-letter queue. As the command is not being processed under sync point control, the broker is not able to retry the command.

Response

Use reason code <*insert_1*> to determine what remedial action is required. If the failure is due to a resource problem (for example, a queue being full), you might find that the problem has already cleared itself. If not, check the application from which the command originated and ensure that it is specifying its reply-to queue correctly. You might find that defining a dead-letter queue to capture the reply message on a subsequent failure will help you with this task.

AMQ5866

Broker command message has been discarded. Reason code <*insert_1*>.

Severity

10 : Warning

Explanation

The IBM WebSphere MQ Publish/Subscribe broker failed to process a publish/subscribe command message, which has now been discarded. The broker will begin to process new command messages again.

Response

Look for previous error messages to indicate the problem with the command message. Correct the problem to prevent the failure from happening again.

AMQ5867

IBM WebSphere MQ Publish/Subscribe broker stream has ended abnormally.

Severity

10 : Warning

Explanation

The broker stream (<*insert_3*>) has ended abnormally for reason <*insert_1*>. The broker will

attempt to restart the stream. If the stream should repeatedly fail, the broker will progressively increase the time between attempts to restart the stream.

Response

Use the reason code <*insert_1*> to investigate why the problem occurred. A reason code of 1 indicates that the stream ended because a command message could not be processed successfully. Look in the error logs for earlier messages to determine the reason why the command message failed. A reason code of 2 indicates that the stream ended because the broker exit could not be loaded. Until the problem with the broker exit has been resolved, the stream will continue to fail.

AMQ5868

User is no longer authorized to subscribe to stream.

Severity

0 : Information

Explanation

The broker has attempted to publish a publication to a subscriber, but the subscriber no longer has browse authority to stream queue (<*insert_4*>). The publication is not sent to the subscriber and his subscription is deregistered. An event publication containing details of the subscription that was removed is published on SYSTEM.BROKER.ADMIN.STREAM. While user ID (<*insert_3*>) remains unauthorized, the broker will continue to deregister subscriptions associated with that user ID.

Response

If the authority of user ID (<*insert_3*>) was intentionally removed, consider removing all of that user IDs subscriptions immediately by issuing an MQCMD_DEREGISTER_SUBSCRIBER command, specifying the MQREGO_DEREGISTER_ALL option on the subscriber's behalf. If the authority was revoked accidentally, reinstate it, but be aware that some, if not all, of the subscriber's subscriptions will have been deregistered by the broker.

AMQ5869

IBM WebSphere MQ Publish/Subscribe broker is checkpointing registrations.

Severity

0 : Information

Explanation

A large number of changes have been made to the publisher and subscriber registrations of stream (<*insert_3*>). These changes are being checkpointed, in order to minimize both stream restart time and the amount of internal queue space being used.

Response

None.

AMQ5870

(Unexpected Error)

Severity

0 : Information

Explanation

N/A

Response

N/A

AMQ5871

(Resource Problem)

Severity

0 : Information

Explanation
N/A

Response
N/A

AMQ5872
(IBM WebSphere MQ Publish/Subscribe broker has a known child)

Severity
0 : Information

Explanation
N/A

Response
N/A

AMQ5873
(IBM WebSphere MQ Publish/Subscribe broker active)

Severity
0 : Information

Explanation
N/A

Response
N/A

AMQ5874
(One or more queues could not be quiesced)

Severity
0 : Information

Explanation
N/A

Response
N/A

AMQ5875
IBM WebSphere MQ Publish/Subscribe broker cannot write a message to the dead-letter queue.

Severity
10 : Warning

Explanation
The broker attempted to put a message to the dead-letter queue (<*insert_3*>) but the message could not be written to the dead-letter queue for reason <*insert_1*>:<*insert_4*>. The message was being written to the dead-letter queue with a reason of <*insert_2*>:<*insert_5*>.

Response
Determine why the message cannot be written to the dead-letter queue. Also, if the message was not deliberately written to the dead-letter queue, for example by a message broker exit, determine why the message was written to the dead-letter queue and resolve the problem that is preventing the message from being sent to its destination.

AMQ5876
A parent conflict has been detected in the IBM WebSphere MQ Publish/Subscribe broker hierarchy.

Severity
20 : Error

Explanation

The broker (<*insert_3*>) has been started, naming this broker as its parent. This broker was started naming broker (<*insert_3*>) as its parent. The broker will send an exception message to broker (<*insert_3*>) indicating that a conflict has been detected. The most likely reason for this message is that the broker topology has been changed while inter-broker communication messages were in transit (for example, on a transmission queue) and that a message relating to the previous broker topology has arrived at a broker in the new topology. This message may be accompanied by an informational FFST including details of the unexpected communication.

Response

If the broker topology has changed and the broker named in the message no longer identifies this broker as its parent, this message can be ignored - for example, if the command "clrmqbrk -m <*insert_3*> -p" was issued. If broker (<*insert_3*>) has been defined as this broker's parent, and this broker has been defined as broker (<*insert_3*>)'s parent, the clrmqbrk or the dltmqbrk commands should be used to resolve the conflict.

AMQ5877

IBM WebSphere MQ Publish/Subscribe broker stream has ended abnormally.

Severity

10 : Warning

Explanation

A broker stream (<*insert_3*>) has ended abnormally for reason <*insert_1*>. The broker recovery routines failed to reset the stream state and the stream cannot be restarted automatically.

Response

Investigate why the stream failed and why the broker's recovery routine could not recover following the failure. Take appropriate action to correct the problem. Depending upon the broker configuration and the nature of the problem it will be necessary to restart either the broker,or both the queue manager and the broker, to make the stream available. If the problem persists save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5878

IBM WebSphere MQ Publish/Subscribe broker recovery failure detected.

Severity

10 : Warning

Explanation

An earlier problem has occurred with the broker, and either a stream has been restarted or the broker has been restarted. The restarted stream or broker has detected that the previous instance of the stream or broker did not clean up successfully and the restart will fail.

Response

Investigate the cause of the failure that caused a stream or broker restart to be necessary, and why the broker or stream was unable to clean up its resources following the failure. When the broker processes with a non trusted routing exit (RoutingExitConnectType=STANDARD), the broker runs in a mode where it is more tolerant of unexpected failures and it is likely that the restart will succeed after a short delay. In the case of a stream restart, the broker will normally periodically retry the failing restart. In the case of a broker restart, it will be necessary to manually retry the broker restart after a short delay. When the broker processes without a routing exit, or with a trusted routine exit (RoutingExitConnectType=FASTPATH), the broker runs in a mode where it is less tolerant of unexpected failures and a queue manager restart will be necessary to resolve this problem. When the broker is running in this mode, it is important that the broker processes are not subjected to unnecessary asynchronous interrupts, for example, kill. If the problem persists, save any generated output files and use either the http://www.ibm.com/

support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ5879

IBM WebSphere MQ Publish/Subscribe broker has been migrated.

Severity

10 : Warning

Explanation

The command was unsuccessful because the MQ Pub/Sub broker at queue manager <*insert_3*> has been migrated. After migration the only command which can be issued against the migrated broker is the dltmqbrk command.

Response

Issue the dltmqbrk command to delete the migrated broker.

AMQ5880

User is no longer authorized to subscribe to stream.

Severity

0 : Information

Explanation

The broker has attempted to publish a publication to a subscriber but the subscriber no longer has altusr authority to stream queue (<*insert_4*>). The publication is not sent to the subscriber and that user IDs subscription is deregistered. An event publication containing details of the subscription that was removed is published on SYSTEM.BROKER.ADMIN.STREAM. While user ID (<*insert_3*>) remains unauthorized, the broker will continue to deregister subscriptions associated with that user ID.

Response

If the authority of user ID (<*insert_3*>) was intentionally removed, consider removing subscriptions immediately by issuing an MQCMD_DEREGISTER_SUBSCRIBER command for the appropriate topics on the subscriber's behalf. If the authority was revoked accidentally, reinstate it, but be aware that some, if not all, of the subscriber's subscriptions will have been deregistered by the broker.

AMQ5881

The IBM WebSphere MQ Publish/Subscribe broker configuration parameter combination <*insert_1*> is not valid.

Severity

20 : Error

Explanation

A combination of Broker stanzas in the queue manager initialization file is not valid. The broker will not operate until this has been corrected.

An combination of (1) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentInputMsg has been set to FALSE. DiscardNonPersistentInputMsg must be set to TRUE when SyncPointIfPersistent is set to TRUE.

An combination of (2) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentResponse has been set to FALSE. DiscardNonPersistentResponse must be set to TRUE when SyncPointIfPersistent is set to TRUE.

An combination of (3) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentPublication has been set to FALSE. DiscardNonPersistentPublication must be set to TRUE when SyncPointIfPersistent is set to TRUE.

Response

Alter the message broker stanzas to comply with the above rules and retry the command.

AMQ5881 (Windows)

The IBM WebSphere MQ Publish/Subscribe broker configuration parameter combination <*insert_1*> is not valid.

Severity

20 : Error

Explanation

A combination of Broker parameters in the broker configuration information is not valid. The broker will not operate until this has been corrected.

An combination of (1) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentInputMsg has been set to FALSE. DiscardNonPersistentInputMsg must be set to TRUE when SyncPointIfPersistent is set to TRUE.

An combination of (2) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentResponse has been set to FALSE. DiscardNonPersistentResponse must be set to TRUE when SyncPointIfPersistent is set to TRUE.

An combination of (3) indicates that SyncPointIfPersistent has been set to TRUE and DiscardNonPersistentPublication has been set to FALSE. DiscardNonPersistentPublication must be set to TRUE when SyncPointIfPersistent is set to TRUE.

Response

Alter the message broker configuration information using the cfgmqbrk tool to comply with the above rules and retry the command.

AMQ5882

IBM WebSphere MQ Publish/Subscribe broker has written a message to the dead-letter queue.

Severity

10 : Warning

Explanation

The broker has written a message to the dead-letter queue (<*insert_3*>) for reason <*insert_1*>:<*insert_5*>. Note. To save log space, after the first occurrence of this message for stream (<*insert_4*>), it will only be written periodically.

Response

If the message was not deliberately written to the dead-letter queue, for example by a message broker exit, determine why the message was written to the dead-letter queue, and resolve the problem that is preventing the message from being sent to its destination.

AMQ5883

IBM WebSphere MQ Publish/Subscribe broker state not recorded.

Severity

10 : Warning

Explanation

The broker state on stream (<*insert_3*>) not recorded while processing a publication outside of sync point. A nonpersistent publication has requested a change to either a retained message or a publisher registration. This publication is being processed outside of sync point because the broker has been configured with the SyncPointIfPersistent option set. A failure has occurred hardening either the publisher registration or the retained publication to the broker's internal queue. All state changes attempted as a result of this publication will be backed-out. Processing of the publication will continue and the broker will attempt to deliver it to all subscribers.

Response

Investigate why the failure occurred. It is probably due to a resource problem occurring on the

broker. The most likely cause is 'queue full' on a broker queue. If your publications also carry state changes, you are advised to send them either as persistent publications or turn off the SyncPointIfPersistent option. In this way, they will be carried out under sync point and the broker can retry them in the event of a failure such as this.

AMQ5884
IBM WebSphere MQ Publish/Subscribe broker control queue is not a local queue.

Severity
10 : Warning

Explanation
IBM WebSphere MQ Publish/Subscribe has detected that the queue 'SYSTEM.BROKER.CONTROL.QUEUE' exists and is not a local queue. This makes the queue unsuitable for use as the control queue of the broker. The broker will terminate immediately.

Response
Delete the definition of the existing queue and, if required, re-create the queue to be of type MQQT_LOCAL. If you do not re-create the queue the broker will automatically create one of the correct type when started.

AMQ5885
Usage: runmqbrk (or strmqbrk) -m QMgrName [-f] [-l logfile]

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ5886
IBM WebSphere MQ Publish/Subscribe broker is being migrated.

Severity
10 : Warning

Explanation
The command cannot be issued at this time because the MQ Pub/Sub broker at queue manager <*insert_3*> is being migrated.

Response
Once migration has commenced then the only command which can be issued against the MQ Pub/Sub broker is the endmqbrk command to cancel the migration. Once the broker has ended if migration did not complete then it can be reattempted using the migmqbrk command again. Alternatively it can be canceled by restarting the broker using the strmqbrk command.

AMQ5887
Migration started for stream <*insert_3*>

Severity
0 : Information

Explanation
Migration of stream <*insert_3*> has started.

Response
None.

AMQ5888
Migration complete for stream <*insert_3*>

Severity
0 : Information

Explanation
All of the state of stream <*insert_3*> has been exported to the WebSphere Brokers broker.

Response
None.

AMQ5889
IBM WebSphere MQ Publish/Subscribe broker has been successfully migrated.

Severity
0 : Information

Explanation
Migration of the broker has completed successfully.

Response
The broker has been migrated. Resources used by it can now be freed by using the dltmqbrk command.

AMQ5890
The migration of the IBM WebSphere MQ Publish/Subscribe broker has failed.

Severity
10 : Warning

Explanation
The IBM WebSphere MQ Publish/Subscribe broker is being migrated. During this migration all persistent state, for example subscriptions, are exported to the WebSphere Brokers broker as a series of messages sent to queue <*insert_3*>. A migration message could not be written to this queue for reason <*insert_1*>.

Response
Use the MQPUT failure code <*insert_1*> to determine why the message cannot be written to the queue. The reason code could indicate that the queue manager is terminating in which case the migmqbrk command will need to be re-issued after the queue manager has restarted. Alternatively there may be a problem with the queue which may need to be rectified before migration can be attempted again.

AMQ5891
IBM WebSphere MQ Publish/Subscribe broker has failed to receive a reply while exporting its state to WebSphere Brokers

Severity
10 : Warning

Explanation
The IBM WebSphere MQ Publish/Subscribe broker is being migrated. During this migration all persistent state, for example subscriptions, are exported to the WebSphere Brokers broker as a series of messages. A reply message for one of the migration messages could not be retrieved from queue <*insert_3*> for reason <*insert_1*>. The migration of the IBM WebSphere MQ Publish/Subscribe broker has failed.

Response
Use the MQGET failure code <*insert_3*> to determine why the reply message could not be received from the reply queue. The reason code could indicate that the queue manager is terminating in which the migmqbrk command will need to be re-issued after the queue manager has restarted. A reason code of 2033 indicates that no reply message was received within a 30 second wait interval. In this case the problem is more likely to have occurred at the WebSphere Brokers broker. Check for error messages issued at the WebSphere Brokers broker.

AMQ5892

Migration of stream <*insert_3*> has failed for reason <*insert_1*>:<*insert_4*>.

Severity

0 : Information

Explanation

Migration of stream <*insert_3*> has failed.

Response

Use reason code <*insert_1*> to investigate the reason for the failure. Once the problem has been resolved, re-issue the migmqbrk command to retry migration.

AMQ5892 (IBM i)

Migration of stream <*insert_3*> has failed.

Severity

0 : Information

Explanation

Migration of stream <*insert_3*> has failed for reason <*insert_1*>:<*insert_4*>.

Response

Use reason code <*insert_1*> to investigate the reason for the failure. Once the problem has been resolved, re-issue the migmqbrk command to retry migration.

AMQ5893

IBM WebSphere MQ Publish/Subscribe broker (<*insert_3*>) cannot be migrated for reason <*insert_1*>:<*insert_5*>.

Severity

20 : Error

Explanation

An attempt has been made to migrate the IBM WebSphere MQ Publish/Subscribe broker (<*insert_3*>) but the request has failed for reason <*insert_1*>:<*insert_5*>.

Response

Determine why the migmqbrk command cannot complete successfully. The message logs for the queue manager might contain more detailed information outlining why the broker cannot be migrated. Resolve the problem that is preventing the command from completing and reissue the migmqbrk command.

AMQ5893 (IBM i)

IBM WebSphere MQ Publish/Subscribe broker cannot be migrated.

Severity

20 : Error

Explanation

An attempt has been made to migrate the broker (<*insert_3*>) but the request has failed for reason <*insert_1*>:<*insert_5*>.

Response

Determine why the migmqbrk command cannot complete successfully. The message logs for the queue manager might contain more detailed information outlining why the broker cannot be migrated. Resolve the problem that is preventing the command from completing and reissue the migmqbrk command.

AMQ5894

IBM WebSphere MQ Publish/Subscribe broker cannot be migrated.

Severity

10 : Warning

Explanation

The IBM WebSphere MQ Publish/Subscribe broker cannot be migrated yet because the state of stream <*insert_3*> is not consistent with respect to related broker <*insert_4*>. While an IBM WebSphere MQ Publish/Subscribe broker is being migrated a check is made to ensure that the state of each stream is consistent with respect to all of the broker's relations. This check has failed because an inconsistency has been detected in the state of stream <*insert_3*> with respect to broker <*insert_4*>. The problem will most likely be of a transient nature, caused because the WebSphere MQ Publish/Subscribe broker has yet to complete processing a recent change to the topology of the broker network. For example, the stream in question may have recently been created or deleted at related broker <*insert_4*> and this broker has yet to complete its processing for this change. Another cause maybe that either this broker, or broker <*insert_4*>, have just been added into the broker network and subscriptions have yet to be exchanged the two brokers. If this is the case then the brokers will be inconsistent with respect to all streams. If no recent topology changes have been made then there maybe a current failure with the propagation of subscriptions to broker <*insert_4*>.

Response

In all cases migration of the IBM WebSphere MQ Publish/Subscribe broker will need to be suspended until the inconsistency has been resolved. You will need to restart the broker using the strmqbrk command so that it can resolve the problem. After a short while, the broker can be ended and migration reattempted. If repeated attempts to migrate the broker all fail with this message then try to resolve the underlying problem. Look for earlier occurrences of message AMQ5826 and follow the guidance given there. In all cases ensure that the channels between the two brokers are running.

AMQ5895

IBM WebSphere MQ Publish/Subscribe broker cannot be migrated.

Severity

10 : Warning

Explanation

A topic has been detected which cannot be exported to the WebSphere Brokers broker. The topic <*insert_3*> cannot be migrated because it contains wildcard characters recognized by the WebSphere Brokers broker. The wildcard characters used by WebSphere Brokers are the '+' and the '#' characters. The state associated with the topic is not migrated and migration of the IBM WebSphere MQ Publish/Subscribe broker fails.

Response

The IBM WebSphere MQ Publish/Subscribe broker cannot be migrated while topic <*insert_3*> is in use. All applications using topics which contain either the '+' or '#' characters will need to be redesigned to use different topic strings. Until the problem has been resolved the IBM WebSphere MQ Publish/Subscribe broker can be restarted as normal using the strmqbrk command.

AMQ5896

Unknown attribute for IBM WebSphere MQ Publish/Subscribe broker configuration parameter GroupId.

Severity

20 : Error

Explanation

The broker has attempted to create stream <*insert_4*> belonging to group <*insert_3*>, this group is unknown.

Response

Modify the attribute for broker configuration parameter GroupId, to a group that exists, or create the group <*insert_3*>.

AMQ5897

Subscription (subname <*insert_5*>, traditional identity <*insert_4*>, topicstring <*insert_3*>) not migrated, reason code <*insert_2*>

Severity

10 : Warning

Explanation

The migration of a subscription has failed and will be skipped (The migration failed with reason code <*insert_2*>). The subscription has topic string is <*insert_3*>, traditional identity <*insert_4*> and subcription name <*insert_5*>.

Response

Either manually migrate this subscription or investigate and fix the problem and perform the migration again.

AMQ5898

Changing parent queue manager cannot be performed during migration.

Severity

20 : Error

Explanation

A different queue manager was supplied with the '-p' parameter to the current parent manager.

Response

Reissue the migration command without the -p option. Once the migration has been performed, use MQSC to alter the queue manager's parent queue manager.

AMQ5900

Usage: migmbbrk [-r] [-o] [-s] [-z] -b BrokerName

Severity

0 : Information

Explanation

This shows the correct usage.

Response

None.

AMQ5901

Migrating Publish/Subscribe ACLs Header.

Severity

0 : Information

Explanation

Migrating Publish/Subscribe ACLs.

From WebSphere Message Broker: <*insert_3*>

To WebSphere MQ Queue Manager: <*insert_4*>

Timestamp: <*insert_5*>

Response

Follow the instructions to migrate ACLs

AMQ5902

Migrating Publish/Subscribe ACLs. No Broker ACLs

Severity

0 : Information

Explanation

The simplest way to migrate to IBM WebSphere MQ is to choose or create a user group with members that are all the user ids that will use publish/subscribe services. Edit the setmqaut command shown here to replace <AllPSUsers> with the group you have chosen. Then execute the resulting command to modify the security attributes of the root MQ topic to be equivalent to WebSphere Brokers

setmqaut -m <*insert_4*> -n SYSTEM.BASE.TOPIC -t topic -g <AllPSUsers> +pub +sub

Response
Follow the instructions to migrate ACLs

AMQ5903
Migrating Publish/Subscribe ACLs. No Negative ACLs

Severity
0 : Information

Explanation

The root of the topic tree in <*insert_3*> has been changed to the same setting that is used by MQ. Furthermore, the topic tree contains only postive ACLs. Therefore it is possible to migrate the ACLs directly from <*insert_3*> to <*insert_4*> as follows.

1. Use the following MQSC commands to create topic objects in the topic tree for <*insert_4*>.

Response
Follow the instructions to migrate ACLs

AMQ5904
Migrating Publish/Subscribe ACLs. MQSC Create Topic

Severity
0 : Information

Explanation

Topic Object Name: <*insert_3*>

Topic String: <*insert_4*>

Response
Follow the instructions to migrate ACLs

AMQ5905
Migrating Publish/Subscribe ACLs. setmqaut

Severity
0 : Information

Explanation

setmqaut -m <*insert_3*> -n <*insert_4*> -t topic <*insert_5*>

Response
Follow the instructions to migrate ACLs

AMQ5906
Migrating Publish/Subscribe ACLs. setmqaut intro

Severity
0 : Information

Explanation
2. Use the following setmqaut commands to create authorisations in <*insert_4*>.

Response
Follow the instructions to migrate ACLs

AMQ5907
Migrating Publish/Subscribe ACLs. Redundant ACLs

Severity
0 : Information

Explanation

The WebSphere Brokers <*insert_3*> has the protection on its root topic set to allow all users to perform all actions (the default). However, there are additional ACLs defined elsewhere in the topic tree that also grant access to named users. These ACLs are redundant because of the setting on the root. You should review the ACLs defined in the broker since they may not be implementing the security that you intend.

Response
Follow the instructions to migrate ACLs

AMQ5908
Migrating Publish/Subscribe ACLs. Manual intervention required.

Severity
0 : Information

Explanation

The WebSphere Brokers <*insert_3*> has an ACL structure that cannot be migrated directly to IBM WebSphere MQ Typically this happens when the broker uses negative ACLs (which appear as "Deny" in the broker tooling) although it can sometimes occur when the root of the topic tree has multiple ACLs. You must review the broker's ACL structure and migrate it manually to <*insert_4*>.

Response
Follow the instructions to migrate ACLs

AMQ5909
Unable to create temporary queue <*insert_3*>.

Severity
20 : Error

Explanation
Unable to create temporary queue <*insert_3*>.

Response
Run the application again with service trace enabled and then contact your IBM support center.

AMQ5910
Unable to open migration log file.

Severity
20 : Error

Explanation
Unable to open migration log file.

Response
The log file is called amqmigmbbrk.log and is created in the current working directory. Determine why this file cannot be created and then re-run this application.

AMQ5911
Unable to delete temporary queue <*insert_3*>.

Severity
20 : Error

Explanation
Unable to delete temporary queue <*insert_3*>.

Response
If the migration log file shows that the application completed successfully then delete queue <*insert_3*> manually. If not, then run the application again with service trace enabled and then contact your IBM support center.

AMQ5912
Unable to open queue <*insert_3*>. Reason code: <*insert_1*>.

Severity
20 : Error

Explanation
Unable to open queue <*insert_3*>. Reason code: <*insert_1*>.

Response
Determine why the application cannot open the queue. Re-running the application while collecting trace may help with this. If necessary, contact your IBM service centre.

AMQ5913
WebSphere Brokers <*insert_3*> is not responding.

Severity
20 : Error

Explanation
WebSphere Brokers <*insert_3*> is not responding.

Response
Check that the WebSphere Brokers <*insert_3*> is started and working normally. If necessary, contact your IBM service centre.

AMQ5914
Unable to read a message from queue <*insert_3*>. Reason code: <*insert_1*>.

Severity
20 : Error

Explanation
Unable to read a message from queue <*insert_3*>. Reason code: <*insert_1*>.

Response
Determine why the application cannot read from the queue. Re-running the application while collecting service trace may help with this. If necessary, contact your IBM service centre.

AMQ5915
Unable to put a message to queue <*insert_3*>. Reason code: <*insert_1*>.

Severity
20 : Error

Explanation
Unable to put a message to queue <*insert_3*>. Reason code: <*insert_1*>.

Response
Determine why the application cannot put to the queue. Re-running the application while collecting service trace may help with this. If necessary, contact your IBM service centre.

AMQ5916
Unable to close queue <*insert_3*>. Reason code: <*insert_1*>.

Severity
20 : Error

Explanation
Unable to close queue <*insert_3*>. Reason code: <*insert_1*>.

Response
Determine why the application cannot close the queue. Re-running the application while collecting trace may help with this. If necessary, contact your IBM service centre.

AMQ5917
Unable to initialise the XML parser.

Severity
20 : Error

Explanation
Unable to initialise the XML parser.

Response
This is an internal error. Re-run the application while collecting service trace, then contact your IBM service centre.

AMQ5918
An XML message from the WebSphere Brokers <*insert_3*> could not be parsed.

Severity
20 : Error

Explanation
An XML message from the WebSphere Brokers <*insert_3*> could not be parsed.

Response
An XML message provided by WebSphere Brokers <*insert_3*> resulted in an error when &MQ tried to parse it. The XML message that caused the problem has been written to <*insert_4*>. The problem occurred in line <*insert_1*> at column <*insert_2*>. Contact your IBM service centre and report this problem.

AMQ5919
The XML parser encountered an error and had to stop.

Severity
20 : Error

Explanation
The XML parser encountered an error and had to stop.

Response
An XML message provided by WebSphere Brokers <*insert_3*> resulted in an error when &MQ tried to parse it. The XML message has been written to <*insert_4*>. Contact your IBM service centre and report this problem.

AMQ5920
Unable to clear temporary queue <*insert_3*>.

Severity
20 : Error

Explanation
Unable to clear temporary queue <*insert_3*>.

Response
Examine the queue and try to clear it manually. If the problem persists then run the application again with service trace enabled and then contact your IBM support center.

AMQ5921
Unable to create UTF-8 transcoder.

Severity
20 : Error

Explanation
Unable to create UTF-8 transcoder. This is an internal error from the XML message parser.

Response
Run the application again with service trace enabled and then contact your IBM support center.

AMQ5922
Unable to migrate a topic string from WebSphere Brokers because it is too long or contains an unrecognised character. The start of the string is <*insert_3*>.

Severity
20 : Error

Explanation
Unable to process a topic string from WebSphere Brokers because it is too long or contains an unrecognized character. The start of the string is <*insert_3*>.

Response
Migrate the topic string manually. (Reviewing the migration log may provide additional information about the source of the problem.)

AMQ5923
Unable to retrieve the CCSID for queue manager <*insert_3*>. Reason code: <*insert_1*>

Severity
20 : Error

Explanation
Unable to retrieve the CCSID for queue manager <*insert_3*>. Reason code: <*insert_1*>

Response
Re-run the application with trace enabled to determine the cause of the problem. If necessary, contact your IBM support centre.

AMQ5924
Duplicate topic object <*insert_3*> already exists.

Severity
20 : Error

Explanation
While attempting to create topic object <*insert_3*> for topic string <*insert_4*> the migration utility found that a topic object of that name already exists and was unable to replace it.

Response
Examine the topic object to determine whether it represents the correct topic string. If it does, then it was probably created by a previous run of this utility and it is safe to either use it as it is, or overwrite it. If not, then the conflict will have to be resolved manually. Further details of this problem are recorded in the migration log file.

AMQ5925
The execution environment for WebSphere Brokers has not been initialised

Severity
20 : Error

Explanation
This utility must be run from a command window that can execute WebSphere Brokers commands and this not the case.

Response

Either run this utility from an WebSphere Brokers command console or manually execute the mqsiprofile command script before running the migration tool.

AMQ5926

Unable to subscribe to topic for migration completion message.

Severity

20 : Error

Explanation

This utility subscribes to topic, <*insert_3*>, to determine whether the pub/sub state for this broker has already been migrated. However the subscription failed with reason code %d.

Response

This is an unexpected error. Contact your IBM support centre

AMQ5927

Migration for this broker has been completed successfully in the past. Since the -z switch was not specified, this attempt will be abandoned.

Severity

0 : Information

Explanation

Migration for this broker has been completed successfully in the past. Since the -z switch was not specified, this attempt will be abandoned.

Response

If the previous successful run produced satisfactory results then there is nothing more to do. If you really do intend to run the migration again then specify the -z switch. You may also want to use the -o switch if you wish to overwrite existing artifacts in the queue manager with ones found during migration.

AMQ5928

Migrating subscription (subname <*insert_5*>, traditional identity <*insert_4*>, topicstring <*insert_3*>) failed when replacing an existing subscription with reason <*insert_2*>

Severity

20 : Error

Explanation

Because the migration command was run with the force flag (-f) it has tried to replace an existing subscription. Replacing the existing subscription failed with reason <*insert_2*>. The subscription has topic string is <*insert_3*>, traditional identity <*insert_4*> and subcription name <*insert_5*>.

Response

Use the migration log to investigate and fix the problem and perform the migration again.

AMQ5929

Migration of a subscription was skipped as a existing subscription exists with the same subname. (The subscription that was not migrated had: subname <*insert_5*>, traditional identity <*insert_4*> and topicstring <*insert_3*>.

Severity

10 : Warning

Explanation

The migration command was run without the force flag (-f). Therefore existing subscriptions are not overwritten. Two subscriptions cannot have the same subname so migration of the subscription was skipped.

Response

If the subscription that has been skipped is still required then either the existing subscription

with the same name can be removed and the migration command then re-run, or the migration command can be re-run with the force option (-f) which will cause any existing subscriptions with the same subname to be migrated.

AMQ5930
Migration of stream <*insert_3*> encountered non-fatal errors, reason <*insert_1*>:<*insert_4*>.

Severity
0 : Information

Explanation
During migration of stream <*insert_3*>, an error occurred but the migration of the stream continued

Response
Use earlier error messages, the migration log, or both to investigate the reason for the failure. Once the problem has been resolved, iissue the migmqbrk command again to retry migration.

AMQ5931
Failed to create topic object for stream <*insert_3*> reason <*insert_1*>

Severity
20 : Error

Explanation
During migration a topic object is created for each stream. Creation of the topic object that corresponds to stream <*insert_3*> failed for reason <*insert_1*>.

Response
Use the migration log to investigate and fix the problem and perform the migration again.

AMQ5932
Migration of security for stream <*insert_3*> failed with reason <*insert_1*>

Severity
20 : Error

Explanation
During migration, security access for the stream is migrated to the corresponding topic object. Migrating the security for stream <*insert_3*> failed for reason <*insert_1*>.

Response
Use the migration log to investigate and fix the problem and perform the migration again.

AMQ5933
Could not open migration log: <*insert_3*>

Severity
20 : Error

Explanation
A log of actions performed during publish/subscribe migration is kept. (Its location can be set using the "-l" command line parameter - currently it is set to <*insert_3*>). The log could not be opened for writing.

Response
Ensure that the file <*insert_3*> can be written to and then rerun the migration. Alternatively rerun the migration specifying a different log file location with the "-l" parameter.

AMQ5934
Could not write to migration log: <*insert_3*>

Severity
20 : Error

Explanation

A log of actions performed during publish/subscribe migration is kept. (Its location can be set using the "-l" command line parameter - currently is is set to <*insert_3*>). The log could not be written to.

Response

Ensure that the file <*insert_3*> can be wriiten to and then rerun the migration. Alternatively rerun the migration specifying a different log file location with the "-l" parameter.

AMQ5935

None of the following subscription properties were encountered during migration

JoinExcl

JoinShared

NoAlter

VariableUserId

SubIdentity

SubName

If your subscriptions do not use these properties then no further action is required. However if you do have subscriptions that rely on these properties then you must upgrade WebSphere Brokers and re-run the migration.

Severity

10 : Warning

Explanation

These properties are only visible to the migration tool if WebSphere Brokers has been upgraded to the most recent fix pack level.

Response

If your subscriptions do not use these properties then no action is required.

However, if any of your subscriptions use any of these properties then you need to upgrade WebSphere Brokers and then re-run the migration process.

AMQ5936

Unable to commit a read from queue <*insert_3*>.

Severity

20 : Error

Explanation

A message was read from queue <*insert_3*> under synch point but the subsequent attempt to commit that read failed.

Response

Re-run the application using the -s switch which will force all intermediate queues to be cleared before they are used. If the problem persists, contact your IBM service centre.

AMQ5937

Duplicate subscription already exists.

Severity

20 : Error

Explanation

While attempting to create a subscription named <*insert_3*> for topic string <*insert_4*> the migration utility found that a subscription of that name already exists and was unable to replace it.

Response

Examine the subscription to determine whether it is correct. If it is, then it was probably created by a previous run of this utility and it is safe to either use it as it is, or overwrite it. If not, then the conflict will have to be resolved manually. Further details of this problem are recorded in the migration log file.

AMQ5938

Unable to make subscription.

Severity

20 : Error

Explanation

A failure occurred while attempting to create a subscription to topic string <*insert_4*> using the subscription name <*insert_3*>. The associated reason code is <*insert_1*>.

Response

Use the reason code shown in the message to determine the cause of the failure and take appropriate action to rectify the problem.

AMQ5939

Unexpected message read from queue <*insert_3*>.

Severity

20 : Error

Explanation

A message read from queue <*insert_3*> was not expected at this stage of migration.

Response

The unexpected message should not have been on the queue. Re-run the application using the -s switch which will force all intermediate queues to be cleared before they are used. If the problem persists, contact your IBM service centre.

AMQ5940

Failed to migrate relations

Severity

20 : Error

Explanation

During migration of the hierarchy relations an error was encountered. See the migration log for more details.

Response

Look at the migration log for details of the error, correct the problem and run the migration command again.

AMQ5941

Unable to allocate a unique name for a subscription point.

Severity

20 : Error

Explanation

The queue manager allocates a unique topic object name for each subscription point up to a maximum of 256 and that limit has been reached. No further subscription points can be migrated to this queue manager. In addition, any artifact that depends on this subscription point, for example, retained publications, will also not be migrated.

Response

If possible reduce the number of subscription points used by the WebSphere Brokers broker that is the source of the migration.

AMQ5942
A userid, <*insert_3*>, supplied by the WebSphere Brokers is not valid

Severity
20 : Error

Explanation
The userid, <*insert_3*>, is not valid for use with the queue manager.

Response
Examine the migration log or product trace to determine why the userid is not valid for this queue manager. If possible, alter the userid that is stored in the broker and re-run the migration step.

AMQ5943
Migration cannot be performed as the IBM WebSphere MQ Publish/Subscribe is currently active

Severity
10 : Warning

Explanation
The runmqbrk (and strmqbrk) commands migrate publish/subscribe data (for example, subscriptions and retained messages) from earlier versions of &MQ. The migration can only be performed when the IBM WebSphere MQ Publish/Subscribe is inactive.

Response
If migration is required, the IBM WebSphere MQ Publish/Subscribe should first be disabled, which can be achieved using the following MQSC: alter qmgr psmode(compat)

AMQ5944
Migration has completed with errors. The IBM WebSphere MQ Publish/Subscribe will need to be started manually

Severity
10 : Warning

Explanation
The migration command has completed but not all data could be migrated. Details of the error(s) can be found in earlier error messages and the migration log.

Response
Examine earlier error messages and review the migration log, then manually perform migration of any remaining data that is still required (or if the problem was transitory by re-running the migration command). Once migration has been completed, the IBM WebSphere MQ Publish/Subscribe can be started by issuing the following MQSC command: alter qmgr psmode(enabled)

AMQ5945
The retained message for topic-string <*insert_3*> on stream <*insert_4*> could not be migrated for reason code <*insert_2*>

Severity
10 : Warning

Explanation
The migration of a retained message has failed and will be skipped (The migration failed with reason code <*insert_2*>). The retained message has topic string <*insert_3*>, on stream <*insert_4*>.

Response
Either manually republish the message for this topic or investigate and fix the problem and perform the migration again.

AMQ5946
The &MQQPUBSUB_short could not be started for reason <*insert_1*>

Severity
20 : Error

Explanation
After the migration, the starting of the &MQQPUBSUB_short could not be performed.

Response
Determine (from the reason) why the &MQQPUBSUB_short could not be started, correct the problem and then manually issue the following MQSC command: ALTER QMGR PSMODE(ENABLED)

AMQ5947
The setting of PSMODE is not COMPAT for queue manager <*insert_1*>

Severity
20 : Error

Explanation
The queue manager property PSMODE must be set to COMPAT for queue manager <*insert_1*> to allow pub/sub migration to happen.

Response
None.

AMQ5948
Some properties of RFH1 format retained messages cannot be retrieved from the broker. If there are RFH1 format retained messages in the broker then you should check that the retained publication that has been migrated to the queue manager is in fact correct.

Severity
10 : Warning

Explanation
Some properties of RFH1 format retained messages cannot be retrieved from the broker. If there are RFH1 format retained messages in the broker then you should check that the retained publication that has been migrated to the queue manager is in fact correct. See the MQ documentation for more details.

Response
Check whether the WMB broker does in fact have retained publications that were published in RFH1 format and, if so, manually migrate them to the queue manager.

AMQ5949
Unable to set environment for mqsistop command.

Severity
20 : Error

Explanation
The migration tool attempts to stop the broker once migration is complete and has to set environment variables in order to do this. The attempt to set one or more of those variables failed.

Response
Review the migration log file or run the migration again with trace turned on to obtain more detail of the reason for the failure.

AMQ5950
Unable to resume an interrupted migration run.

Severity
20: Error

Explanation

The migration tool found that a previous run had been interrupted. It normally attempts to resume that migration run from the point where it was interrupted, but on this occasion was unable to do so because the interruption occurred while processing multiple subIdentities for a subscription.

Response

Run the migration again with the -s switch turned on to prevent resumption of the previous run and also with the -o switch to force existing definitions in the queue manager to be overwritten by the definitions brought from the broker.

AMQ5960

Distributed pub/sub command processor stopping because of errors.

Severity

20 : Error

Explanation

A severe error, as reported in the preceding messages, occurred during distributed pub/sub command processing. The pub/sub command processor was unable to continue and terminates.

Response

Correct the problem reported in the preceding messages.

AMQ5961

Distributed pub/sub publication processor stopping because of errors.

Severity

20 : Error

Explanation

A severe error, as reported in the preceding messages, occurred during distributed pub/sub publication processing. The pub/sub publication processor was unable to continue and terminates.

Response

Correct the problem reported in the preceding messages.

AMQ5962

Distributed pub/sub proxy-subscription fan out process is stopping because of errors.

Severity

20 : Error

Explanation

A severe error, as reported in the preceding messages, occurred during distributed pub/sub proxy-subscription fan out. The pub/sub proxy-subscription fan out process was unable to continue and terminates.

Response

Correct the problem reported in the preceding messages.

AMQ5963

Queued Pub/Sub Daemon Unavailable.

Severity

20 : Error

Explanation

The Distributed publish/subscribe process was unable to contact the queued pub/sub Daemon. If there is a problem with the Daemon, this should be highlighted in preceding messages. Hierarchical connections will not be further processed until the problem is rectified.

Response
Correct the problem reported in the preceding messages. When the Daemon becomes available, it may be necessary to perform a REFRESH QMGR TYPE(PROXYSUB) to resync subscriptions.

AMQ5964
Pub/sub hierarchy connected.

Severity
0 : Information

Explanation
A pub/sub hierarchy connection has been established with child or parent queue manager <*insert_3*>.

Response
None.

AMQ5965
Pub/sub hierarchy disconnected.

Severity
0 : Information

Explanation
A pub/sub hierarchy connection has ended with child or parent queue manager <*insert_3*>.

Response
None.

AMQ5966
A previous publication is being incorrectly processed again.

Severity
30 : Severe error

Explanation
A publication, previously processed by this queue manager, has been received. This message will not be published again and will be processed according to the message's report options. Additional messages may be written if this publication is sent to the dead-letter queue. This is caused by an invalid configuration of a hierarchy and a pub/sub cluster.

Response
Correct the configuration to remove the loop. Check the message properties in the dead-letter queue to determine the route taken.

AMQ5967
Unable to deliver proxy subscription to queue manager <*insert_3*>. Reason code: <*insert_1*>.

Severity
20 : Error

Explanation
Unable to deliver proxy subscription to queue manager <*insert_3*>. Reason code: <*insert_1*>. This may result in subscriptions not receiving publications from <*insert_3*>.

Response
Correct the configuration to allow proxy subscriptions to be delivered to <*insert_3*>. When the problem has been resolved, it will be necessary to perform a REFRESH QMGR TYPE(PROXYSUB) to resynchronize subscriptions.

AMQ5972
Distributed Pub/Sub fanout request put failed.

Severity
20 : Error

Explanation

Unable to place subscription fanout request to the Distributed publish/subscribe fanout request queue <*insert_3*>. The associated reason code is <*insert_1*>.

Response

Correct the problem reported in the preceding messages. When the problem has been resolved, it may be necessary to perform a REFRESH QMGR TYPE(PROXYSUB) to resync subscriptions.

AMQ5979

Proxy subscription from <*insert_3*> rejected because PSCLUS(DISABLED).

Severity

10 : Warning

Explanation

Queue manager attribute PSCLUS has been set to DISABLED to indicate that Publish/Subscribe activity is not expected between queue managers in this cluster. However, a cluster subscription has been sent to this queue manager over a channel from <*insert_3*>. The proxy subscription request will be ignored and no subscription locally registered.

Response

If you need to enable publish/subscribe clustering, alter the PSCLUS attribute on all queue managers in the cluster to ENABLED. You might also need to issue REFRESH CLUSTER and REFRESH QMGR commands as detailed in the PSCLUS documentation. If you are not using publish/subscribe clusters you should delete the clustered topic object, and ensure that PSCLUS is DISABLED on all queue managers.

AMQ5980

Distributed Pub/Sub proxy subscription re-synchronization occurred at startup.

Severity

0 : Information

Explanation

The Distributed publish/subscribe process was unable to determine that the proxy subscription state was consistent across shutdown and restart so a re-synchronization with remote queue managers has been performed. This is usually seen when a queue manager was not quiesced cleanly on the previous shutdown or when the system was particularly busy.

Response

None.

AMQ5981

Disabling Publish/Subscribe when participating in a Publish/Subscribe Cluster.

Severity

10 : Warning

Explanation

This queue manager is a member of a Publish/Subscribe Cluster but Publish/Subscribe has been disabled. Other queue managers within the cluster will continue to send publications and proxy subscriptions to this queue manager. They will accumulate on the Publish/Subscribe Cluster system queues and will not be processed until Publish/Subscribe is enabled. If these queues become full channel failure may occur, which will affect the operation of Publish/Subscribe on other queue managers in the cluster. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are sent to this queue manager from other queue managers within the cluster.

Response

Enable Publish/Subscribe by setting PSMODE to ENABLED or COMPAT with the ALTER QMGR command, and then issue the REFRESH QMGR TYPE(PROXYSUB) command to resynchronize subscriptions.

AMQ5982

Disabling Queued Publish/Subscribe while participating in a Publish/Subscribe Hierarchy.

Severity

10 : Warning

Explanation

This queue manager is a member of a Publish/Subscribe hierarchy but Queued Publish/Subscribe has been disabled. Any parent-child relations within the Publish/Subscribe hierarchy will continue to send publications and proxy subscriptions to this queue manager. They will accumulate on the Queued Publish/Subscribe system queues and will not be processed until Queued Publish/Subscribe is enabled. If the Queued Publish/Subscribe system queues become full channel failure may occur, which will affect the operation of Publish/Subscribe on parent-child relations sending messages to this queue manager. This will also affect the delivery of other messages, unrelated to Publish/Subscribe, that are to be delivered using the same channels.

Response

Enable Queued Publish/Subscribe by setting PSMODE to ENABLED with the ALTER QMGR command. When Queued Publish/Subscribe has been restarted, use the DISPLAY PUBSUB ALL command to confirm this has completed, and then issue the REFRESH QMGR TYPE(PROXYSUB) command to resynchronize subscriptions.

AMQ6000-6999: Common services

AMQ6004

An error occurred during IBM WebSphere MQ initialization or ending.

Severity

30 : Severe error

Explanation

An error was detected during initialization or ending of IBM WebSphere MQ The IBM WebSphere MQ error recording routine has been called.

Response

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6005 (IBM i)

An error occurred during IBM WebSphere MQ startup.

Severity

30 : Severe error

Explanation

An attempt to start the storage monitor process (job QMQM in subsystem QSYSWRK) was unsuccessful.

Response

Check the joblog for this job and for the QMQM job for possible reasons for failure, correct the error and try the command again. If the problem is not resolved, a problem may have been logged. Use WRKPRB to record the problem identifier, and to save the QPSRVDMP, QPJOBLOG, and QPDSPJOB files. Save any generated output files and use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6015

The operating system is either too busy or has insufficient resources to complete a system request.

Severity

30 : Severe error

Explanation

A system request <*insert_3*> was rejected by the operating system with return code <*insert_1*>. IBM WebSphere MQ retried the request, but it continued to fail. This failure may indicate that the operating system is either too busy or has insufficient resources to complete the request.

Response

Investigate whether the system is constrained by the workload on this system or by the workload on a server that it is using, and reduce the workload.

AMQ6024

Insufficient resources are available to complete a system request.

Severity

30 : Severe error

Explanation

A system request was rejected by the operating system because insufficient resources are available to complete the request. Use any previous FFSTs, error log messages or, on Windows, system event log messages, to determine which resource is insufficient.

Response

Investigate whether the system has been configured in accordance with the documentation and increase the necessary resource to allow the system request to complete successfully.

AMQ6025

Program not found.

Severity

30 : Severe error

Explanation

IBM WebSphere MQ is unable to start program <*insert_3*> because it was not found.

Response

Check the program name is correctly specified and rerun the program.

AMQ6026

A resource shortage prevented the creation of a IBM WebSphere MQ process.

Severity

30 : Severe error

Explanation

An attempt to create an IBM WebSphere MQ process was rejected by the operating system due to a process limit (either the number of processes for each user or the total number of processes running system wide), or because the system does not have the resources necessary to create another process.

Response

Investigate whether a process limit is preventing the creation of the process and if so why the system is constrained in this way. Consider raising this limit or reducing the workload on the system.

AMQ6035

IBM WebSphere MQ failed, no storage available.

Severity
30 : Severe error

Explanation
An internal function of the product attempted to obtain storage, but there was none available.

Response
Stop the product and restart it. If this does not resolve the problem, save any generated output files and use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ6037
IBM WebSphere MQ was unable to obtain enough storage.

Severity
20 : Error

Explanation
The product is unable to obtain enough storage. The product's error recording routine may have been called.

Response
Stop the product and restart it. If this does not resolve the problem see if a problem has been recorded. If a problem has been recorded, use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6047
Conversion not supported.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to convert string data tagged in CCSID <*insert_1*> to data in CCSID <*insert_2*>.

Response
Check the IBM WebSphere MQ Application Programming Reference Appendix and the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6048
DBCS error

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to convert string data due to a DBCS error. Conversion is from CCSID <*insert_1*> to CCSID <*insert_2*>.

Response
Check the IBM WebSphere MQ Application Programming Reference Appendix and the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6049
DBCS-only string not valid.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to convert string data in CCSID <*insert_1*> to data in CCSID <*insert_2*>. Message descriptor data must be in single-byte form. CCSID <*insert_2*> is a DBCS-only CCSID.

Response
Check the CCSID of your job or system and change it to one supporting SBCS or mixed character sets. Refer to the IBM WebSphere MQ Application Programming Reference Appendix and the appropriate National Language Support publications for character sets and CCSIDs supported.

AMQ6050
CCSID error.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to convert string data in CCSID <*insert_1*> to data in CCSID <*insert_2*>.

Response
Check the IBM WebSphere MQ Application Programming Reference Appendix and the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6051
Conversion length error.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to convert string data in CCSID <*insert_1*> to data in CCSID <*insert_2*>, due to an input length error.

AMQ6052
Conversion length error.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to convert string data in CCSID <*insert_1*> to data in CCSID <*insert_2*>.

AMQ6053
CCSID error

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to convert string data in CCSID <*insert_1*> to data in CCSID <*insert_2*>.

Response
One of the CCSIDs is not supported by the system. Check the IBM WebSphere MQ Application

Programming Reference Appendix and the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6064
An internal IBM WebSphere MQ error has occurred.

Severity
30 : Severe error

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6088 (IBM i)
An internal IBM WebSphere MQ error has occurred.

Severity
40 : Stop Error

Explanation
An internal error occurred when API call <*insert_3*> was made.

Response
Use WRKPRB to record the problem identifier, and to save the QPSRVDMP, QPJOBLOG, and QPDSPJOB files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6089 (IBM i)
IBM WebSphere MQ was unable to display an error message.

Severity
30 : Severe error

Explanation
An attempt to display an error message was unsuccessful. This may be because the AMQMSG message file could not be found. The message identifier is <*insert_3*>.

Response
Check that the library list is set up correctly to access the AMQMSG message file. If a change is necessary, rerun the failing application and record the error message. If you are unable to resolve the problem, save any generated output files and use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ6090
IBM WebSphere MQ was unable to display an error message <*insert_6*>.

Severity
0 : Information

Explanation

IBM WebSphere MQ has attempted to display the message associated with return code hexadecimal <*insert_6*>. The return code indicates that there is no message text associated with the message. Associated with the request are inserts <*insert_1*> : <*insert_2*> : <*insert_3*> : <*insert_4*> : <*insert_5*>.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6091

An internal IBM WebSphere MQ error has occurred.

Severity

0 : Information

Explanation

Private memory has detected an error, and is abending due to <*insert_3*>. The error data is <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6092 (Windows)

Manual conversion required for CCSID: <*insert_1*>

Severity

0 : Information

Explanation

CCSID <*insert_1*> exists in new format but could not be reconciled against your old format.

Response

Manually edit CCSID entry <*insert_1*> in conv\table\ccsid.tbl if you wish to retain your old conversion. For assistance call your Service Representative.

AMQ6100

An internal IBM WebSphere MQ error has occurred.

Severity

0 : Information

Explanation

IBM WebSphere MQ has detected an error, and is abending due to <*insert_3*>. The error data is <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a

solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6103 (IBM i)
IBM WebSphere MQ job submission error.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to submit job <*insert_3*>.

AMQ6107
CCSID not supported.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to convert string data in CCSID <*insert_1*> to data in CCSID <*insert_2*>, because one of the CCSIDs is not recognized.

Response
Check the IBM WebSphere MQ Application Programming Reference Appendix and the appropriate National Language Support publications to see if the CCSIDs are supported by your system.

AMQ6109
An internal IBM WebSphere MQ error has occurred.

Severity
30 : Severe error

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6110
An internal IBM WebSphere MQ error has occurred.

Severity
30 : Severe error

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6112 (IBM i)
IBM WebSphere MQ CCSID <*insert_1*> is using a default value.

Severity
10 : Warning

Explanation
When initializing IBM WebSphere MQ, no valid job CCSID was found, so the CCSID used is the default 37. This warning message will be issued until a valid CCSID has been set correctly.

Response
Set the job CCSID.

AMQ6114 (IBM i)
An internal IBM WebSphere MQ error has occurred.

Severity
30 : Severe error

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
Use WRKPRB to record the problem identifier, and to save the QPSRVDMP, QPJOBLOG, and QPDSPJOB files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6115
An internal IBM WebSphere MQ error has occurred.

Severity
10 : Warning

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6118
An internal IBM WebSphere MQ error has occurred (<*insert_1*>)

Severity
40 : Stop Error

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6119
An internal IBM WebSphere MQ error has occurred (<*insert_3*>)

Severity
40 : Stop Error

Explanation
IBM WebSphere MQ detected an unexpected error when calling the operating system. The IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6120
An internal IBM WebSphere MQ error has occurred.

Severity
40 : Stop Error

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6121
An internal IBM WebSphere MQ error has occurred.

Severity
40 : Stop Error

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
IBM WebSphere MQ has detected a parameter count of <*insert_1*> that is not valid. Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6122
An internal IBM WebSphere MQ error has occurred.

Severity
40 : Stop Error

Explanation
An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response
IBM WebSphere MQ has detected parameter <*insert_1*> that is not valid, having value <*insert_2*><*insert_3*>. Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web

page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6125
An internal IBM WebSphere MQ error has occurred.

Severity
40 : Stop Error

Explanation
An internal error has occurred with identifier <*insert_1*>. This message is issued in association with other messages.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6134 (IBM i)
Trace continues in buffer

Severity
0 : Information

AMQ6135 (IBM i)
Stopping early trace

Severity
0 : Information

AMQ6136 (IBM i)
Stopping early trace <*insert_3*> system time

Severity
0 : Information

AMQ6137 (IBM i)
Resuming MQI trace

Severity
0 : Information

AMQ6138 (IBM i)
Resuming MQI trace <*insert_3*> system time

Severity
0 : Information

AMQ6139 (IBM i)
Stopping MQI trace

Severity
0 : Information

AMQ6140 (IBM i)
Stopping MQI trace <*insert_3*> system time

Severity
0 : Information

AMQ6141 (IBM i)
Starting MQI trace

Severity
0 : Information

AMQ6142 (IBM i)
Starting MQI trace <*insert_3*> system time

Severity
0 : Information

AMQ6143 (IBM i)
IBM WebSphere MQ function stack

Severity
0 : Information

AMQ6144 (IBM i)
No stack available

Severity
0 : Information

AMQ6145 (IBM i)
Terminating MQI trace

Severity
0 : Information

AMQ6146 (IBM i)
Entering end job processing

Severity
0 : Information

AMQ6147 (IBM i)
Terminating MQI trace <*insert_3*> system time

Severity
0 : Information

AMQ6148
An internal IBM WebSphere MQ error has occurred.

Severity
0 : Information

Explanation
IBM WebSphere MQ has detected an error, and is abending due to <*insert_3*>. The error data is <*insert_1*>.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6150 (Windows)
IBM WebSphere MQ semaphore is busy.

Severity
10 : Warning

Explanation

IBM WebSphere MQ was unable to acquire a semaphore within the normal timeout period of <*insert_1*> minutes.

Response

IBM WebSphere MQ will continue to wait for access. If the situation does not resolve itself and you suspect that your system is locked then investigate the process which owns the semaphore. The PID of this process will be documented in the accompanying FFST.

AMQ6150 (IBM i)

IBM WebSphere MQ resource <*insert_3*> busy.

Severity

30 : Severe error

Explanation

IBM WebSphere MQ was unable to access an IBM WebSphere MQ object within the normal timeout period of <*insert_1*> minutes.

Response

IBM WebSphere MQ will continue to wait for access. Ensure that all jobs using IBM WebSphere MQ are released. If the situation persists, quiesce the queue manager.

AMQ6151 (IBM i)

IBM WebSphere MQ resource <*insert_3*> released.

Severity

30 : Severe error

Explanation

An IBM WebSphere MQ resource, for which another process has been waiting, for a period of over <*insert_1*> minutes has been released.

Response

No recovery is needed.

AMQ6152 (IBM i)

IBM WebSphere MQ failed to end commitment control while attempting to quiesce a queue manager.

Severity

30 : Severe error

Explanation

IBM WebSphere MQ failed to end commitment control whilst quiescing queue manager <*insert_3*>.

Response

There are one or more active resources under commitment control. Use the Work with Job (WRKJOB) command with the OPTION(*CMTCTL) parameter to display the active resources under commitment control. Check the job log for previously issued messages.

AMQ6153 (IBM i)

The attempt to quiesce queue manager <*insert_3*> failed

Severity

30 : Severe error

Explanation

The attempt to quiesce queue manager <*insert_3*> was unsuccessful

Response
Check the job log for previously issued messages. If the quiesce was issued with the *CNTRLD option, re-issue the command with the *IMMED option. If a low TIMEOUT retry delay was used, re-issue the request with a higher value.

AMQ6154 (IBM i)
Queue manager <*insert_3*> has been quiesced.

Severity
0 : Information

Explanation
The queue manager has been successfully quiesced.

Response
None.

AMQ6158 (IBM i)
SBCS CCSID not found.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ is unable to find an SBCS CCSID which corresponds to mixed DBCS-SBCS CCSID <*insert_1*>.

Response
Check the CCSID of your job or system and check it has a SBCS equivalent. Refer to the National Language Support Planning Guide for character sets and CCSIDs supported. If the CCSID used does have an SBCS equivalent, save the job log containing this message and use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ6159 (IBM i)
IBM WebSphere MQ job submission error.

Severity
30 : Severe error

Explanation
IBM WebSphere MQ for IBM i is unable to release job <*insert_3*>.

Response
Contact you System Administrator to remove job <*insert_3*>. Ensure you have *JOBCTL authority and try again.

AMQ6160
EXPLANATION:

Severity
0 : Information

AMQ6161
ACTION:

Severity
0 : Information

AMQ6162
An error has occurred reading an INI file.

Severity
20 : Error

Explanation
An error has occurred when reading the MQS.INI file or a queue manager QM.INI file.

Response
If you have been changing the INI file content check and correct the change. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6162 (Tandem)
An error has occurred reading an INI file.

Severity
20 : Error

Explanation
An error has occurred when reading the MQSINI file or a queue manager QMINI file.

Response
If you have been changing the INI file content check and correct the change. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6162 (Windows)
An error occurred when reading the configuration data.

Severity
20 : Error

Explanation
An error has occurred when reading the configuration data.

Response
If you have changed the configuration data, check and correct the change. If you have not changed the configuration data, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6163
An error has occurred locking an INI file.

Severity
10 : Warning

Explanation
An error has occurred locking the MQS.INI file or a queue manager QM.INI file.

Response
If you have been changing the INI file permissions check and correct the change. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem

identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6163 (Tandem)
An error has occurred locking an INI file.

Severity
10 : Warning

Explanation
An error has occurred locking the MQSINI file or a queue manager QMINI file.

Response
If you have been changing the INI file permissions check and correct the change. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6163 (Windows)
An error has occurred locking the configuration data.

Severity
10 : Warning

Explanation
An error has occurred locking the configuration data.

Response
If you have changed the the registry permissions, check and correct the change. If you have not changed the registry, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6164
An expected stanza in an INI file is missing or contains errors.

Severity
10 : Warning

Explanation
An expected stanza is missing from the MQS.INI file or a queue manager QM.INI file or the stanza contains errors.

Response
If you have been changing the INI file content check and correct the change.

AMQ6164 (Tandem)
An expected stanza in an INI file is missing or contains errors.

Severity
10 : Warning

Explanation
An expected stanza is missing from the MQSINI file or a queue manager QMINI file or the stanza contains errors.

Response
If you have been changing the INI file content check and correct the change.

AMQ6164 (Windows)
An expected stanza in the configuration data is missing or contains errors.

Severity
10 : Warning

Explanation
An expected stanza is missing from the configuration data or the stanza contains errors.

Response
If you have changed the configuration data, check and correct the change.

AMQ6165
Unable to access an INI file.

Severity
10 : Warning

Explanation
Access to the MQS.INI file or a queue manager QM.INI file is denied.

Response
If you have been changing the INI file permissions check and correct the change.

AMQ6165 (Tandem)
Unable to access an INI file.

Severity
10 : Warning

Explanation
Access to the MQSINI file or a queue manager QMINI file is denied.

Response
If you have been changing the INI file permissions check and correct the change.

AMQ6165 (Windows)
Unable to access the configuration data.

Severity
10 : Warning

Explanation
Access to the configuration data is denied.

Response
If you have changed the configuration data permissions, check and correct the changes.

AMQ6166
An INI file is missing.

Severity
20 : Error

Explanation
The MQS.INI file or a queue manager QM.INI file is missing.

Response
If you have been changing the INI file recover the previous file and retry the operation.

AMQ6166 (Tandem)
An INI file is missing.

Severity
20 : Error

Explanation
The MQSINI file or a queue manager QMINI file is missing.

Response
If you have been changing the INI file recover the previous file and retry the operation.

AMQ6166 (Windows)
An entry in the configuration data is missing.

Severity
20 : Error

Explanation
A required entry in the configuration data is missing.

Response
If you have changed the configuration data, recover the previous configuration data and retry the operation.

AMQ6172
No codeset found for current locale.

Severity
20 : Error

Explanation
No codeset could be determined for the current locale. Check that the locale in use is supported.

Response
None.

AMQ6173
No CCSID found for codeset <*insert_3*>.

Severity
20 : Error

Explanation
Codeset <*insert_3*>. has no supported CCSID. Check that the locale in use is supported. CCSIDs can be added by updating the file /var/mqm/conv/table/ccsid.tbl.

Response
None.

AMQ6174
The library <*insert_3*> was not found. The queue manager will continue without this module.

Severity
0 : Information

Explanation
The dynamically loadable library <*insert_3*> was not found.

Response
Check that the file exists and is either fully qualified or is in the appropriate directory.

AMQ6174 (UNIX and Linux)
The dynamically loadable shared library <*insert_3*> was not found. The system returned error number <*insert_2*> and error message <*insert_4*>. The queue manager will continue without this module.

Severity
0 : Information

Explanation
This message applies to UNIX and Linux systems. The shared library <*insert_3*> was not found.

Response
Check that the file exists, and is either fully qualified or is in the appropriate director, also check the file access permissions.

AMQ6174 (IBM i)
The library was not found.

Severity
0 : Information

Explanation
The dynamically loadable file <*insert_3*> was not found. The queue manager will continue without this module.

Response
Check that the file exists and is either fully qualified or is in the appropriate directory.

AMQ6175 (AIX)
The system could not dynamically load the shared library <*insert_3*>. The system returned error number <*insert_2*> and error message <*insert_4*>. The queue manager will continue without this module.

Severity
20 : Error

Explanation
This message applies to AIX systems. The shared library <*insert_3*> failed to load correctly due to a problem with the library.

Response
Check the file access permissions and that the file has not been corrupted.

AMQ6175 (UNIX and Linux)
The system could not dynamically load the shared library <*insert_3*>. The system returned error message <*insert_4*>. The queue manager will continue without this module.

Severity
20 : Error

Explanation
This message applies to UNIX and Linux systems. The shared library <*insert_3*> failed to load correctly due to a problem with the library.

Response
Check the file access permissions and that the file has not been corrupted.

AMQ6175 (Windows)
The system could not dynamically load the library <*insert_3*>. The system returned error message <*insert_4*>. The queue manager will continue without this module.

Severity
20 : Error

Explanation
This message applies to Windows NT and Windows 2000 systems only. The dynamically loadable file <*insert_3*> failed to load correctly due to an internal error. The IBM WebSphere MQ error recording routine has been called.

Response

Check that the file has not been corrupted then use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6177 (Windows)

An internal IBM WebSphere MQ error has occurred.

Severity

40 : Stop Error

Explanation

An error has been detected, and the IBM WebSphere MQ error recording routine has been called.

Response

Details of the error have been stored at <*insert_3*>. A synopsis is given in the data section below. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6179

The system could not find symbol <*insert_5*> in the dynamically loaded library <*insert_3*>. The system returned error number <*insert_2*> and error message <*insert_4*>.

Severity

20 : Error

Explanation

The library <*insert_3*> does not contain symbol <*insert_5*> or it has not been exported.

Response

Check that symbol name <*insert_5*> is correct and has been exported from the library.

AMQ6179 (UNIX and Linux)

The system could not find the symbol <*insert_5*> in the dynamically loaded shared library <*insert_3*>. The system returned error message <*insert_4*>.

Severity

20 : Error

Explanation

This message applies to UNIX and Linux systems. The shared library <*insert_3*> does not contain symbol <*insert_5*> or it has not been exported.

Response

Check that symbol name <*insert_5*> is correct and has been exported from the library.

AMQ6180 (Windows)

Default conversion not supported.

Severity

30 : Severe error

Explanation

IBM WebSphere MQ is unable to convert string data tagged in CCSID <*insert_1*> to data in CCSID <*insert_2*>.

Response

Check the default CCSIDs specified in the ccsid.tbl file and make sure that conversion is supported between these CCSIDs.

AMQ6182

Error found in line <*insert_1*> of ccsid.tbl

Severity

30 : Severe error

Explanation

Line <*insert_1*> contains and error. The content of the line is <*insert_3*>. Processing continues but the line in error is ignored.

Response

Correct the line and rerun the program or command giving this message.

AMQ6183

An internal IBM WebSphere MQ error has occurred.

Severity

10 : Warning

Explanation

An error has been detected, and the IBM WebSphere MQ error recording routine has been called. The failing process is process <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6184

An internal IBM WebSphere MQ error has occurred on queue manager <*insert_3*>.

Severity

10 : Warning

Explanation

An error has been detected, and the IBM WebSphere MQ error recording routine has been called. The failing process is process <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6184 (IBM i)

An internal IBM WebSphere MQ error has occurred.

Severity

10 : Warning

Explanation

An internal IBM WebSphere MQ error has occurred on queue manager <*insert_3*> and the IBM WebSphere MQ error recording routine has been called. The failing process is process <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6187

User is not authorized for RestrictedMode queue manager.

Severity

40 : Stop Error

Explanation

All users must be in the RestrictedMode application_group.

AMQ6188 (AIX)

The system could not dynamically load the shared library <*insert_3*> as the entry point to the library, symbol 'MQStart', could not be located within the library. The queue manager will continue without this library.

Severity

20 : Error

Explanation

This message applies to AIX systems. The shared library <*insert_3*> failed to load correctly due to a problem with the library.

Response

Check that the entry point to the library, symbol 'MQStart', exists and has been exported from the library.

AMQ6188 (UNIX and Linux)

The system could not dynamically load the shared library <*insert_3*> as the entry point to the library, symbol 'MQStart', could not be located within the library. The system returned error message <*insert_4*>. The queue manager will continue without this library.

Severity

20 : Error

Explanation

This message applies to UNIX and Linux systems. The shared library <*insert_3*> failed to load correctly due to a problem with the library.

Response

Check that the entry point to the library, symbol 'MQStart', exists and has been exported from the library.

AMQ6188 (Windows)

The system could not dynamically load the library <*insert_3*> due to a problem with the dll. The errno was <*insert_1*>. The queue manager will continue without this module.

Severity

20 : Error

Explanation

This message applies to Windows NT and Windows 2000 systems only. The dynamically loadable file <*insert_3*> failed to load correctly due to a problem with the dll.

Response

Check that the dll is in the correct place with the correct file permissions etc. and has not been corrupted.

AMQ6190 (Windows)
Program <*insert_3*> not found.

Severity
30 : Severe error

Explanation
The program <*insert_3*> cannot be found.

Response
Check that the program specified is available on your system. If the program name is not fully qualified, ensure that the PATH environment variable includes the directory where the program is located.

AMQ6191 (Windows)
Program <*insert_3*> failed to start, return code <*insert_1*>.

Severity
30 : Severe error

Explanation
The program <*insert_3*> was invoked, but failed to start. The failure reason code is <*insert_1*>.

Response
Check that the program specified is available on your system, and that sufficient system resources are available. Where applicable, verify that the user is authorized to run the program.

AMQ6192 (Windows)
IBM IBM WebSphere MQ Utilities

Severity
0 : Information

AMQ6193 (Windows)
The registry entry <*insert_3*> was not found.

Severity
20 : Error

Explanation
IBM WebSphere MQ for Windows NT and Windows 2000 sets the registry entry <*insert_3*> when the product is installed, but the entry is now missing.

Response
If the registry has been edited, restore the previous version. If the product is newly installed, check whether the installation was successful, and reinstall the product if necessary.

AMQ6196
An error has occurred whilst processing a temporary INI file <*insert_3*>

Severity
20 : Error

Explanation
An error has occurred when creating a backup of an INI file. The backup file <*insert_4*> already exists

Response
You may have created a backup of the INI file with the name <*insert_4*>, or an earlier operation may have failed. Move or delete the file <*insert_4*> and reattempt the operation. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to

see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ6207 (AIX)

Failed to attach shared memory segment as Segment table is Full.

Severity

20 : Error

Explanation

IBM WebSphere MQ has attempted to attach a memory segment but was unable to do so because all available segment areas are in use. 32 bit programs on AIX may attach up to a maximum of 10 shared memory segments. If the application has modified the data area layout, for example by reserving more of the address space for the program heap, this maximum number may be further reduced.

Response

Examine the needs of your application to see if the number of attached segments can be reduced. Alternatively by building your application as a 64bit program the limit of 10 shared memory segments is removed.

AMQ6209

An unexpected asynchronous signal (<*insert_1*> : <*insert_3*>) has been received and ignored.

Severity

10 : Warning

Explanation

Process <*insert_2*> received an unexpected asynchronous signal and ignored it. This has not caused an error but the source of the signal should be determined as it is likely that the signal has been generated externally to IBM WebSphere MQ

Response

Determine the source of the signal and prevent it from recurring.

AMQ6212

Failed to load Library <*insert_3*> as C++ environment is not initialised.

Severity

20 : Error

Explanation

An attempt was made to load the identified C++ shared library. However, the attempt failed because the C++ environment has not been initialized for the current process.

Response

Ensure the application is linked with the appropriate C++ runtime environment.

AMQ6218 (AIX)

EXTSHM variable detected with unrecognised value <*insert_3*> and has been reset to <*insert_4*>.

Severity

20 : Error

Explanation

Processes that access the internal queue manager control blocks must use the AIX Extended Shared Memory model, and while one such process was starting, IBM WebSphere MQ detected that the EXTSHM variable was set but did not contain an appropriate value. This value has been reset and the process will continue with the new setting.

Response

No further action is required. To prevent this message being issued in future, correct the value of the EXTSHM variable in your environment.

AMQ6224 (Tandem)
The environment variable have not been set up correctly.

Severity
10 : Warning

Response
Check that the environment variables correspond to the configuration file.

AMQ6230
Message <*insert_3*> suppressed <*insert_1*> times in the last <*insert_4*> seconds.

Severity
10 : Warning

Explanation
Message <*insert_3*> was issued <*insert_2*> times in the last <*insert_4*> seconds but only the first instance of the message was written to the log. The suppressed messages may have included differing message arguments.

Response
If you wish to see all occurrences of this message you should alter the definition of the SuppressMessage attribute in the Queue Manager configuration.

AMQ6232 (UNIX and Linux)
Operating System userid <*insert_3*> not found.

Severity
20 : Error

Explanation
A request was made to the operating system to lookup the details of the identified userid but the request failed.

Response
Using the operating system supplied tools check for the existence of the identified userid, and if missing then re-create it.

AMQ6233 (UNIX and Linux)
Operating System authorisation group <*insert_3*> not found.

Severity
20 : Error

Explanation
A request was made to the operating system to lookup the details of the identified group but the request failed.

Response
Using the operating system supplied tools check for the existence of the identified group, and if missing then re-create it.

AMQ6234 (UNIX and Linux)
Unknown Queue Manager name specified.

Severity
20 : Error

Explanation
An invalid Queue Manager name <*insert_3*> was specified in the parameters to the command.

Response
Reissue the command specifying a valid Queue Manager name.

AMQ6235 (UNIX and Linux)
Directory <*insert_3*> missing.

Severity
20 : Error

Explanation
The identified directory is missing.

Response
Reissue the command selecting the option to create missing directories.

AMQ6236 (UNIX and Linux)
Missing directory <*insert_3*> has been created.

Severity
20 : Error

Explanation
The identified directory was missing but has been created.

Response
None

AMQ6237 (UNIX and Linux)
File <*insert_3*> missing.

Severity
20 : Error

Explanation
The identified file is missing.

Response
Reissue the command selecting the option to create missing files.

AMQ6238 (UNIX and Linux)
Missing file <*insert_3*> has been created.

Severity
20 : Error

Explanation
The identified file was missing but has been created.

Response
None

AMQ6239 (Windows, UNIX and Linux)
Permission denied attempting to access filesystem location <*insert_3*>.

Severity
20 : Error

Explanation
An attempt to query the filesystem object identified failed because the command issued did not have authority to access the object.

Response
Check the authority on the object and of the user executing the command and reissue the command.

AMQ6240 (UNIX and Linux)
You must be an operating system superuser to run this command.

Severity
20 : Error

Explanation
In irder to run this command you must be logged on as a user with superuser privelages.

Response
Log in as an appropriate user and reissue the command.

AMQ6241 (UNIX and Linux)
The filesystem object <*insert_3*> is a symbolic link.

Severity
20 : Error

Explanation
While checking the filesystem, an object was found which is a symbolic link.

Response
This is not an error however you should verify that the symbolic link is expected and that the destination of the symbolic link is correct.

AMQ6242 (UNIX and Linux)
Incorrect ownership for <*insert_3*>. Current(<*insert_1*>) Expected(<*insert_2*>)

Severity
20 : Error

Explanation
The filesystem object <*insert_3*> is owned by the user with uid <*insert_1*> when it was expected to be owned by the user with uid <*insert_2*>.

Response
Correct the ownership using operating system commands or reissue the command selecting the option to fix the incorrect owership.

AMQ6243 (UNIX and Linux)
Incorrect group ownership for <*insert_3*>. Current(<*insert_1*>) Expected(<*insert_2*>)

Severity
20 : Error

Explanation
The filesystem object <*insert_3*> is owned by the group with gid <*insert_1*> when it was expected to be owned by the group with gid <*insert_2*>.

Response
Correct the ownership using operating system commands or reissue the command selecting the option to fix the incorrect owership.

AMQ6244 (UNIX and Linux)
Incorrect permissions on object <*insert_3*>. Current(<*insert_4*>) Expected(<*insert_5*>)

Severity
20 : Error

Explanation
The filesystem object <*insert_3*> has the wrong file permissions.

Response
Correct the ownership using operating system commands or reissue the command selecting the option to fix the incorrect owership.

AMQ6245 (UNIX and Linux)
Error executing system call <*insert_3*> on file <*insert_4*> error <*insert_2*>.

Severity
20 : Error

Explanation
The execution of the system call <*insert_3*> on file <*insert_4*> failed and the error code <*insert_2*> was returned.

Response
Investigate the cause of the failure using the operating system error code <*insert_1*> and reissue the command.

AMQ6251 (UNIX and Linux)
The system could not dynamically load the shared library <*insert_3*>. The queue manager will continue without this module.

Severity
20 : Error

Explanation
This message applies to UNIX and Linux systems. The shared library <*insert_3*> failed to load as it is probably a <*insert_1*>-bit library, a <*insert_2*>-bit library is required. Note that IBM WebSphere MQ tried to find a <*insert_2*>-bit library named either <*insert_4*> or <*insert_5*>, but failed. The following message gives details of the original failure.

Response
Supply the name of a <*insert_2*>-bit library.

AMQ6252 (UNIX and Linux)
The system could not dynamically load the shared library <*insert_3*>. The queue manager will continue without this module.

Severity
20 : Error

Explanation
This message applies to UNIX and Linux systems. The shared library <*insert_3*> failed to load as it is probably a <*insert_1*>-bit library, a <*insert_2*>-bit library is required. Note that IBM WebSphere MQ found and loaded a <*insert_2*>-bit library named <*insert_4*> however this also failed to load with the system returning error message <*insert_5*>. The following message gives details of the original failure.

Response
Supply the name of a <*insert_2*>-bit library.

AMQ6253 (UNIX and Linux)
The system could not dynamically load the shared library <*insert_3*>. The queue manager will continue without this module.

Severity
20 : Error

Explanation
This message applies to UNIX and Linux systems. The shared library <*insert_3*> failed to load as it is probably a <*insert_1*>-bit library, a <*insert_2*>-bit library is required. Note that IBM WebSphere MQ attempted to locate and load a <*insert_2*>-bit library named either of these: <*insert_4*>. The first library failed to load as it also is probably a <*insert_1*>-bit library, the second library is a <*insert_2*>-bit library, however this also failed to load with the system returning error message <*insert_5*>. The following message gives details of the original failure.

Response
Supply the name of a <*insert_2*>-bit library.

AMQ6254 (UNIX and Linux)

The system could not dynamically load the shared library <*insert_3*>, library <*insert_4*> has been used instead.

Severity

0 : Information

Explanation

This message applies to UNIX and Linux systems. The shared library <*insert_3*> failed to load as it is probably a <*insert_1*>-bit library, a <*insert_2*>-bit library is required. Note that IBM WebSphere MQ has sucessfully located and loaded a <*insert_2*>-bit library named <*insert_4*>.

Response

Supply the name of a <*insert_2*>-bit library or put the library (alternatively a symbolic link can be used) in the appropriate place: 32-bit libraries in /var/mqm/exits; 64-bit libraries in /var/mqm/exits64.

AMQ6255 (UNIX and Linux)

The system could not dynamically load the shared library <*insert_3*>. The queue manager will continue without this module.

Severity

20 : Error

Explanation

This message applies to UNIX and Linux systems. The shared library <*insert_3*> failed to load as it is probably a <*insert_1*>-bit library, a <*insert_2*>-bit library is required. The following message gives details of the original failure.

Response

Supply the name of a <*insert_2*>-bit library.

AMQ6256 (Windows)

The system could not dynamically load the shared library <*insert_3*>. The queue manager will continue without this module.

Severity

20 : Error

Explanation

This message applies to Windows systems. The shared library <*insert_3*> failed to load as it is probably a <*insert_1*>-bit library, a <*insert_2*>-bit library is required. Note that IBM WebSphere MQ tried to find a <*insert_2*>-bit library named <*insert_4*>, but failed. The following message gives details of the original failure.

Response

Supply the name of a <*insert_2*>-bit library.

AMQ6257

Message suppression enabled for message numbers (<*insert_3*>).

Severity

0 : Information

Explanation

The message contain's a list of message id's for which entries repeated within the <*insert_1*> suppression interval will be suppressed.

Response

If you wish to see all occurrences of these messages you should alter the definition of the SuppressMessage attribute in the Queue Manager configuration.

AMQ6258

Message exclusion enabled for message numbers (<*insert_3*>).

Severity
0 : Information

Explanation
The message contain's a list of message id's which have been excluded. Requests to write these messages to the error log will be discarded.

Response
If you wish to see instances of these messages you should alter the definition of the ExcludeMessage attribute in the Queue Manager configuration.

AMQ6259
Message <*insert_3*> cannot be <*insert_4*>.

Severity
10 : Warning

Explanation
Message <*insert_3*> cannot be excluded or suppressed but was specified in the ExcludeMessage or SuppressMessage configuration for the Queue Manager. The Queue Manager will continue however the request to suppress or exclude this message will be ignored.

Response
Update the Queue Manager configuration to remove the specified message identifier.

AMQ6260
Help Topic not found

Severity
10 : Warning

Explanation
The requested help topic could not be located.

For further assistance, refer to the IBM WebSphere MQ manuals.

Response
Ensure that the IBM WebSphere MQ InfoCenter is installed.

AMQ6261 (UNIX and Linux)
An exception occurred trying to dynamically load shared library <*insert_3*>. The queue manager will continue without this module.

Severity
20 : Error

Explanation
This message applies to UNIX and Linux systems. Exception number <*insert_1*> name <*insert_4*>, occurred trying to dynamically load shared library <*insert_3*>.

Response
Check the shared library has not been corrupted. If the shared library contains any initializer functions, ensure these are not causing the problem and that they conform to the expected function prototype.

AMQ6261 (Windows)
An exception occurred trying to load DLL <*insert_3*>. The queue manager will continue without this module.

Severity
20 : Error

Explanation
This message applies to Windows systems only. Exception number <*insert_1*> error <*insert_4*>, occurred trying to load DLL <*insert_3*>.

Response

Check the DLL has not been corrupted. If the DLL contains any initializer functions, ensure these are not causing the problem and that they conform to the expected function prototype.

AMQ6263

Usage: dspmqras [-t CollectionType]

Severity

20 : Error

Response

None.

AMQ6266 (Windows)

Error <*insert_1*> occurred accessing shared trace data, <*insert_3*>

Severity

30 : Severe error

Explanation

The IBM WebSphere MQ common services module needs to access an area of named shared memory so that various functions, including trace, can be co-ordinated between all processes on a machine or session.

For a server install, this area should have been created by the IBM WebSphere MQ services process (amqsvc.exe) and is thus shared globally, on a client-only install, or where the IBM WebSphere MQ services are not running, it should be created for this session only.

This failure implies that the named shared memory (normally mqm.SHRSEG.0) has been created by another process on the system in such a way that access to it from IBM WebSphere MQ processes is denied.

Response

Investigate which process on the machine has created the named shared memory and, if it is an IBM WebSphere MQ process or IBM WebSphere MQ application investigate why the permissions have been set to disallow others to connect.

If the process that created this area is not related to IBM WebSphere MQ, investigate why it has created this specifically named area.

AMQ6271

Detected 64-bit JVM, but not using the Resource Recovery Services adapter

Severity

30 : Severe error

Explanation

The only zOS adapter supported in the 64-bit mode is the Resource Recovery Services adapter

Response

Do not specify the com.ibm.mq.adapter system property

AMQ6272

com.ibm.mq.adapter set to <*insert_0*>, which is invalid

Severity

30 : Severe error

Explanation

The adapter is not valid in this environment

Response

Set the com.ibm.mq.adapter to a valid value

AMQ6276

group name <*insert_3*> size <*insert_1*> is too long to be used for <*insert_4*>.

Severity

20 : Error

Explanation

<*insert_4*> has not been authorised for use by the groupname <*insert_3*>. This will not affect users who are members of group mqm.

Response

Save any generated output files and use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ6277

function name <*insert_5*> returned <*insert_1*> when creating a SID for group <*insert_3*> while creating object '$4'.

Severity

20 : Error

Explanation

<*insert_4*> has not been authorised for use by the groupname <*insert_3*>. This will not affect users who are members of group mqm.

Response

Save any generated output files and use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ6280

Usage: **amqxdbg** ([-x] (-i pid[.tid] | -p program_name) | -s)

Severity

00 : Information

Explanation

The user provided an incorrect set of arguments to the **amqxdbg** command.

```
-i - Request a program FDC from the process identified by
     'pid' and 'tid'.
-p - Request a program FDC from the process identified by the
     supplied program name. To match more than one program name
     the wildcard character '*' may be used at the end of the
     'program_name' specification.
-x - Delete the entry identified by the -i or -p parameters
-s - Show the status of debug entries
```

Response

Re-issue the command using the appropriate arguments.

AMQ6281

Debug entry defined.

Severity

00 : Information

Explanation

The **amqxdbg** command completed successfully and a debug entry was added.

Response

None.

AMQ6282
Debug entry removed.

Severity
00 : Information

Explanation
The **amqxdbg** command completed successfully and a debug entry was removed.

Response
None.

AMQ6283
Debug entry not found.

Severity
20 : Error

Explanation
The debug entry identified was not found and could not removed.

Response
None.

AMQ6284
Debug entry could not be defined. The limit on the number of entries has been reached.

Severity
20 : Error

Explanation
The **amqxdbg** command attempted to add a debug entry but could not becuase the limit on the number of entries which can be defined was reached.

Response
Use the '-x' option to remove debug entries which are no longer required and re-issue the command.

AMQ6285
Process <*insert_1*> does not exist.

Severity
20 : Error

Explanation
The **amqxdbg** command attempted to add a debug entry but could not because the process with process identifier <*insert_1*> is not running.

Response
Check the supplied process identifier and re-issue the command.

AMQ6286
The filesystem at location <*insert_3*> is read-only.

Severity
20 : Error

Explanation
An attempt to write to the filesystem failed because it is read-only. Likely causes are that you specified the location incorrectly or the filesystem has been incorrectly configured.

Response
Identify where the location was specified and check it is correct. Check that the filesystem has been configured correctly.

AMQ6287
IBM WebSphere MQ V<*insert_5*>.

Severity
00 : Information

Explanation
IBM WebSphere MQ system information:

```
Host Info         :- <insert_3>
Installation      :- <insert_4>
Version           :- <insert_5>
```

Response
None.

AMQ6290
Unknown installation <*insert_3*> detected.

Severity
20 : Error

Explanation
When executing program <*insert_4*>, IBM WebSphere MQ detected that, due to the configuration of the environment, resources were loaded from <*insert_3*>. MQ could not determine the installation name for these resources. The program cannot complete successfully while the program is executing using resources from an unknown installation.

Response
Configure the environment so that all resources required by program <*insert_4*> are loaded from an correctly installed installation.

AMQ6290 (UNIX)
Unknown installation path <*insert_3*> detected.

Severity
20 : Error

Explanation
When executing program <*insert_4*>, MQ detected that its resources were loaded from <*insert_3*>, MQ could not determine from <*insert_5*> the installation name and identifier for these resources. The program cannot complete successfully while the program is executing using resources from an unknown installation.

Response
Check that <*insert_5*> exists and has an Installation entry with 'Path=<*insert_3*>'. If '<*insert_5*> has been corrupted, run **crtmqinst -r** to reconstruct the file.

AMQ6291
Error <*insert_1*> occurred during IBM WebSphere MQ process initialization.

Severity
20 : Error

Explanation
An unexpected error was encountered while initializing the process. The process will terminate immediately. The error was <*insert_1*>. The MQ error recording routine may have been called.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard any files until the problem has been resolved.

AMQ6292
The queue manager is associated with a different installation.

Severity
20 : Error

Explanation
A command was issued which attempted to connect to a queue manager, but the installation that the command was issued from does not match the installation that the queue manager is associated with. The attempt to connect failed.

Response
Reissue the command from the installation that the queue manager is associated with.

AMQ6293
Cannot create symbolic link as a file with the name <*insert_3*> already exists. Error Number: <*insert_1*>.

Severity
20 : Error

Explanation
An attempt was made to create a symbolic link with the name <*insert_3*> but the symbolic link could not be created as a file already exists with the same name.

Response
Verify if the file named <*insert_3*> as been created in error. If so, remove before reissuing the command. The Error Number may give more details about the cause of the failure.

AMQ6294
Failed to create symbolic link with the name <*insert_3*>. Error Number: <*insert_1*>.

Severity
20 : Error

Explanation
An attempt was made to create a symbolic link with the name <*insert_3*> but the symbolic link could not be created.

Response
The Error Number for the failure may give details about why the symbolic link could not be created. Correct the problem before reissuing the command.

AMQ6295
Unable to remove symbolic link with the name <*insert_3*>. Error Number: <*insert_1*.

Severity
20 : Error

Explanation
An attempt was made to remove a symbolic link with the name <*insert_3*> but the symbolic link could not be removed.

Response
The Error Number for the failure may give details about why the symbolic link could not be removed. Correct the problem before reissuing the command.

AMQ6296
Cannot remove file <*insert_3*> as it is not a symbolic link.

Severity
20 : Error

Explanation

An attempt was made to remove a symbolic link with the name <*insert_3*> but it was not removed because the file was not a symbolic link.

Response

Check the definition of the symbolic link and, if incorrect, remove the file before reissuing the command.

AMQ6297

Symbolic link with the name <*insert_3*> cannot be removed. Target <*insert_4*> does not match the expected target <*insert_5*>.

Severity

20 : Error

Explanation

An attempt was made to remove a symbolic link with the name <*insert_3*> but it was not removed because the target of the symbolic link <*insert_4*>does not match the expected target <*insert_5*>.

Response

Check the definition of the symbolic link and, if incorrect, remove the symbolic link manually before reissuing the command.

AMQ6299

An error occurred while creating or checking the directory structure for the queue manager.

Severity

40 : Stop Error

Explanation

During creation, startup or deletion of the queue manager, an error occurred while creating or checking a file or directory. The queue manager could not access the path <*insert_3*>.

Response

None.

AMQ6666 (IBM i)

Required IBM WebSphere MQ system profile(s) can not be accessed.

Severity

40 : Stop Error

Explanation

The required IBM WebSphere MQ system profile(s) QMQM, QMQMADM, or both are not found or have been disabled. IBM WebSphere MQ cannot continue processing the command without the profiles existing and enabled on the system. The major error code is <*insert_3*>, the minor error code is <*insert_4*>. The major error codes and their meanings are as follows: *DISABLED - The user profile has been disabled. *PWDEXP - The password for the user profile has expired . *EXIST - The user profile does not exist. If none of these error codes are shown the major error code contains the exception identifier. The minor error code identifies the user profile which cannot be accessed.

Response

Check that both QMQM and QMQMADM profiles exist and are both enabled using the DSPUSRPRF command, or contact the IBM WebSphere MQ system administrator.

AMQ6708

A disk full condition was encountered when formatting a new log file in location <*insert_3*>.

Severity

20 : Error

Explanation

The queue manager attempted to format a new log file in directory <*insert_3*>. The drive or file system containing this directory did not have sufficient free space to contain the new log file.

Response

Increase the amount of space available for log files and retry the request.

AMQ6708 (IBM i)

A disk full condition was encountered when formatting a new log file.

Severity

20 : Error

Explanation

The queue manager attempted to format a new log file in directory <*insert_3*>. The drive or file system containing this directory did not have sufficient free space to contain the new log file.

Response

Increase the amount of space available for log files and retry the request.

AMQ6709

The log for the Queue manager is full.

Severity

20 : Error

Explanation

This message is issued when an attempt to write a log record is rejected because the log is full. The queue manager will attempt to resolve the problem.

Response

This situation may be encountered during a period of unusually high message traffic. However, if you persistently fill the log, you may have to consider enlarging the size of the log. You can either increase the number of log files by changing the values in the queue manager configuration file. You will then have to stop and restart the queue manager. Alternatively, if you need to make the log files themselves bigger, you will have to delete and re-create the queue manager.

AMQ6710

Queue manager unable to access directory <*insert_3*>.

Severity

20 : Error

Explanation

The queue manager was unable to access directory <*insert_3*> for the log. This could be because the directory does not exist, or because the queue manager does not have sufficient authority.

Response

Ensure that the directory exists and that the queue manager has authority to read and write to it. Ensure that the LogPath attribute in the queue manager's configuration file matches the intended log path.

AMQ6767

Log file <*insert_3*> could not be opened for use.

Severity

20 : Error

Explanation

Log file <*insert_3*> could not be opened for use. Possible reasons include the file being missing, the queue manager being denied permission to open the file or the contents of the file being incorrect.

Response

If the log file was required to start the queue manager, ensure that the log file exists and that the queue manager is able to read from and write to it. If the log file was required to re-create an object from its media image and you do not have a copy of the required log file, delete the object instead of recreating it.

AMQ6774

Log file <*insert_3*> did not contain the requested log record.

Severity

20 : Error

Explanation

Log file <*insert_3*> does not contain the log record with an LSN that is <*insert_4*>. This is because the log file numbers have wrapped and the log file name <*insert_3*> has been reused by a newer file. Once a log file name has been reused, it is not possible to access the data in the previous versions of the file to use this name. The operation which requested this log record cannot be completed.

AMQ6782

The log file numbers have wrapped.

Severity

0 : Information

Explanation

Each log file formatted is assigned a number which makes up part of its file name. The numbers are allocated sequentially and consist of seven digits giving a maximum of 10 million different log file names. Once all available numbers have been allocated, the queue manager again starts allocating numbers starting from zero. Once a file number has been re-allocated, you can no longer access data in the previous log files allocated the same number. The file numbers wrapped at log sequence number <*insert_3*>.

Response

You should periodically take media images of all IBM WebSphere MQ objects. You must ensure that media images of all objects which you may need to re-create do not span more than 10 million log files.

AMQ6901 (IBM i)

IBM WebSphere MQ for IBM i

AMQ6902 (IBM i)

IBM WebSphere MQ for IBM i - Samples

AMQ6903 (IBM i)

Installation or uninstallation failed, IBM WebSphere MQ resources are still active.

Severity

30 : Severe error

Explanation

An attempt to install or uninstall IBM WebSphere MQ was unsuccessful because IBM WebSphere MQ resources from a previous installation of IBM WebSphere MQ are still active. This failure may indicate that a queue manager from a previous installation of IBM WebSphere MQ is still running or has active jobs.

Response

Ensure that all queue managers from previous installations of IBM WebSphere MQ have been quiesced, and that the QMQM subsystem is not active using the WRKSBS and ENDSBS commands. Refer to the installation section in the IBM WebSphere MQ for IBM i Quick Beginnings publication for further details.

AMQ6904 (IBM i)
Installation of IBM WebSphere MQ for IBM i failed due to previous release installed.

Explanation
Some releases of IBM WebSphere MQ for IBM i require migration before a later release can be installed.

Response

If you wish to retain your current IBM WebSphere MQ information you must step through the migration process - see the Quick Beginnings Manual.

If you do not wish to retain your current IBM WebSphere MQ information remove the current version of IBM WebSphere MQ before retrying the install.

AMQ6905 (IBM i)
Found <*insert_3*> new IBM WebSphere MQ jobs to end, and <*insert_4*> IBM WebSphere MQ jobs currently ending.

Severity
0 : Information

Explanation
Jobs with locks on library QMQM are ended so that IBM WebSphere MQ may be deleted or updated.

Response
None.

AMQ6906 (IBM i)
<*insert_3*> jobs still ending.

Severity
40 : Stop Error

Explanation
Jobs report state of 'already being deleted' after timeout.

Response
If system is heavily loaded wait and reissue the command CALL QMQM/AMQIQES4 to try to delete jobs using IBM WebSphere MQ resources. If this message is issued again, issue the command WRKOBJLCK for library QMQM to see which jobs have not been deleted, and end them manually.

AMQ6907 (IBM i)
All IBM WebSphere MQ pre-requisite PTFs on OS/400 programs are installed.

Severity
0 : Information

Explanation
None.

Response
None.

AMQ6908 (IBM i)
IBM WebSphere MQ pre-requisite PTF <*insert_4*> for program <*insert_3*> is not installed.

Severity
40 : Stop Error

Explanation

PTF <*insert_3*>-<*insert_4*> is not installed on system in state 'Permanently applied' 'Temporarily applied' or 'Superseded'. IBM WebSphere MQ installation will proceed, but you must install the PTF before starting IBM WebSphere MQ

Response

Use the command GO CMDPTF to display commands to order and apply the required PTF <*insert_3*>-<*insert_4*>..

AMQ6909 (IBM i)

User space recovery failed, IBM WebSphere MQ is running.

Severity

30 : Severe error

Explanation

An attempt to recover user space was unsuccessful because IBM WebSphere MQ was running.

Response

Quiesce IBM WebSphere MQ for IBM i and try again. See the section on "Quiescing IBM WebSphere MQ" in the IBM WebSphere MQ for IBM i Quick Beginnings.

AMQ6910 (IBM i)

The attempt to quiesce the queue manager failed.

Severity

30 : Severe error

Explanation

The attempt to quiesce the queue manager was unsuccessful because the current job has locks on library QMQM.

Response

Sign off the current job, sign on and attempt to quiesce the queue manager again. See the section on "Quiescing IBM WebSphere MQ" in the IBM WebSphere MQ for IBM i Quick Beginnings.

AMQ6911 (IBM i)

IBM WebSphere MQ quiesce is performing a RCDMQMIMG. There may be some delay before completion.

Severity

0 : Information

Explanation

IBM WebSphere MQ quiesce is performing a Record Object Image (RCDMQMIMG) for all objects. There may be some delay until before completion.

Response

None.

AMQ6912 (IBM i)

IBM WebSphere MQ Java Messaging and Web Services

AMQ6913 (IBM i)

IBM WebSphere MQ Java Messaging and Web Services

AMQ6914 (IBM i)

Apply PTF failed, IBM WebSphere MQ resources are still active.

Severity

30 : Severe error

Explanation

An attempt to apply PTFs to a IBM WebSphere MQ installation was unsuccessful because IBM WebSphere MQ resources are still active. This failure may indicate that one or more queue

managers have not been fully quiesced, some IBM WebSphere MQ resources have not been released, some IBM WebSphere MQ jobs are still running or a IBM WebSphere MQ subsystem is still active.

Response

Ensure that all queue managers have been fully quiesced, using the ENDMQM command with ENDCCTJOB(*YES). Ensure that all IBM WebSphere MQ subsystems (including the QMQM subsystem) are not active using the WRKSBS and ENDSBS commands. Repeat the apply PTF action. Note - Delete Licensed Program (DLTLICPGM) is not a circumvention for this condition, because the same checks which are listed as a possible cause, will be made before deleting a IBM WebSphere MQ installation.

AMQ6915 (IBM i)

Remove PTF failed, IBM WebSphere MQ resources are still active.

Severity

30 : Severe error

Explanation

An attempt to remove PTFs from a IBM WebSphere MQ installation was unsuccessful because IBM WebSphere MQ resources are still active. This failure may indicate that one or more queue managers have not been fully quiesced, some IBM WebSphere MQ resources have not been released, some IBM WebSphere MQ jobs are still running or a IBM WebSphere MQ subsystem is still active.

Response

Ensure that all queue managers have been fully quiesced, using the ENDMQM command with ENDCCTJOB(*YES). Ensure that all IBM WebSphere MQ subsystems (including the QMQM subsystem) are not active using the WRKSBS and ENDSBS commands. Repeat the remove PTF action. Note - Delete Licensed Program (DLTLICPGM) is not a circumvention for this condition, because the same checks which are listed as a possible cause, will be made before deleting a IBM WebSphere MQ installation.

AMQ6988

yes

Severity

0 : Information

AMQ6988 (IBM i)

Yes

AMQ6989

no

Severity

0 : Information

AMQ6989 (IBM i)

No

AMQ6992 (IBM i)

Program <*insert_3*> parameter error.

Severity

40 : Stop Error

Explanation

IBM WebSphere MQ for IBM i program <*insert_3*> has an incorrect number of parameters, or an error in the parameter value.

Response

Display the job log, using the DSPJOBLOG command, for more information on the problem.

AMQ6993 (IBM i)
Program <*insert_3*> ended abnormally.

Severity
40 : Stop Error

Explanation
A IBM WebSphere MQ for IBM i program, <*insert_3*>, is ending abnormally.

Response
Display the job log, using the DSPJOBLOG command, for information why the job or subsystem ended abnormally. Correct the error and retry the request.

AMQ6994 (Windows)
5724-H72 (C) Copyright IBM Corp. 1994, 2008. ALL RIGHTS RESERVED.

Severity
0 : Information

Explanation
None.

Response
None.

AMQ6995 (IBM i)
xcsFFST has been called; take a look at the job log.

Severity
0 : Information

AMQ6998 (IBM i)
An internal IBM WebSphere MQ error has occurred.

Severity
40 : Stop Error

Explanation
IBM WebSphere MQ for IBM i is diagnosing an unexpected error.

Response
Save the job log, and use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ6999 (IBM i)
An internal IBM WebSphere MQ error has occurred.

Severity
0 : Information

Explanation
IBM WebSphere MQ has experienced an internal failure, from which it could not recover.

Response
Use WRKPRB to check if a problem has been created. If one has, record the problem identifier, and save the QPSRVDMP, QPJOBLOG, and QPDSPJOB files. If a problem has not been created, save the job log. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7000-7999: WebSphere MQ product

AMQ7001

The location specified for creation of the queue manager is not valid.

Severity

40 : Stop Error

Explanation

The directory under which queue managers are to be created is not valid. It might not exist, or there might be a problem with authorization.

Response

The location is specified in the machine-wide ini file. Correct the file and submit the request again.

AMQ7001 (Windows)

The location specified for the creation of the queue manager is not valid.

Severity

40 : Stop Error

Explanation

The directory under which the queue managers are to be created is not valid. It might not exist, or there might be a problem with authorization.

Response

The location is specified in the configuration data. Correct the configuration data and submit the request again.

AMQ7002

An error occurred manipulating a file.

Severity

40 : Stop Error

Explanation

An internal error occurred while trying to create or delete a queue manager file. It is likely that the error was caused by a disk having insufficient space, or by problems with authorization to the underlying file system.

Response

Identify the file that caused the error, using problem determination techniques. For example check if there are any FFST files, which might identify the queue manager file causing the error. This error might also be caused if users have created, renamed or deleted that file. Correct the error in the file system and submit the request again.

AMQ7002 (Windows)

An error occurred manipulating a file.

Severity

40 : Stop Error

Explanation

An internal error occurred while trying to create or delete a queue manager file.

In the case of a failure to delete a file a common reason for this error is that a non MQ process, such as the windows explorer or a virus checker, is accessing the file. In the case where the object that cannot be deleted is a directory then a non MQ process might be accessing a file within the directory or one of its subdirectories.

It is also possible that the error was caused by a disk having insufficient space, or by problems with authorization to the underlying file system.

Response

Identify the file that caused the error, using problem determination techniques. For example check if there are any FFST files, which might identify the queue manager file causing the error. This error might also be caused if users have created, renamed or deleted that file. Correct the error in the file system and submit the request again.

AMQ7005

The queue manager is running.

Severity

40 : Stop Error

Explanation

You tried to perform an action that requires the queue manager stopped, however, it is currently running. You probably tried to delete or start a queue manager that is currently running.

Response

If the queue manager should be stopped, stop the queue manager and submit the failed command again.

AMQ7006

Missing attribute <*insert_5*> on stanza starting on line <*insert_1*> of ini file <*insert_3*>.

Severity

20 : Error

Explanation

The <*insert_4*> stanza starting on line <*insert_1*> of configuration file <*insert_3*> is missing the required <*insert_5*> attribute.

Response

Check the contents of the file and retry the operation.

AMQ7006 (Windows)

Missing attribute <*insert_5*> from configuration data.

Severity

20 : Error

Explanation

The <*insert_4*> stanza in the configuration data is missing the required <*insert_5*> attribute.

Response

Check the contents of the configuration data and retry the operation.

AMQ7008

The queue manager already exists.

Severity

40 : Stop Error

Explanation

You tried to create a queue manager that already exists.

Response

If you specified the wrong queue manager name, correct the name and submit the request again.

AMQ7010

The queue manager does not exist.

Severity

40 : Stop Error

Explanation

You tried to perform an action against a queue manager that does not exist. You might have specified the wrong queue manager name.

Response

If you specified the wrong name, correct it and submit the command again. If the queue manager should exist, create it, and then submit the command again.

AMQ7011

The queue manager files have not been completely deleted.

Severity

40 : Stop Error

Explanation

While deleting the queue manager, an error occurred deleting a file or directory. The queue manager might not have been completely deleted.

Response

Follow problem determination procedures to identify the file or directory and to complete deletion of the queue manager.

AMQ7012

The specified trigger interval is not valid.

Severity

40 : Stop Error

Explanation

You specified a value for the trigger interval that is not valid. The value must be not less than zero and not greater than 999 999 999.

Response

Correct the value and resubmit the request.

AMQ7013

There is an error in the name of the specified dead-letter queue.

Severity

40 : Stop Error

Explanation

You specified a name for the dead-letter queue that is not valid.

Response

Correct the name and resubmit the request.

AMQ7014

There is an error in the name of the specified default transmission queue.

Severity

40 : Stop Error

Explanation

You specified a name for the default transmission queue that is not valid.

Response

Correct the name and submit the command again.

AMQ7015

There is an error in the maximum number of open object handles specified.

Severity

40 : Stop Error

Explanation

You specified a value for the maximum number of open object handles to be allowed that is not valid. The value must be not less than zero and not greater than 999 999 999.

Response

Correct the value and submit the command again.

AMQ7016

There is an error in the maximum number of uncommitted messages specified.

Severity

40 : Stop Error

Explanation

You specified a value for the maximum number of uncommitted messages to be allowed that is not valid. The value must be not less than 1 and not greater than 999 999 999.

Response

Correct the value and submit the command again.

AMQ7017

Log not available.

Severity

40 : Stop Error

Explanation

The queue manager was unable to use the log. This could be due to a log file being missing or damaged, or the log path to the queue manager being inaccessible.

Response

Ensure that the LogPath attribute in the queue manager configuration file is correct. If a log file is missing or otherwise unusable, restore a backup copy of the file, or the entire queue manager.

AMQ7018

The queue manager operation cannot be completed.

Severity

20 : Error

Explanation

An attempt has been made to perform an operation on a queue manager. Resources required to perform the operation are not available.

AMQ7019

An error occurred while creating or checking the directory structure for the queue manager.

Severity

40 : Stop Error

Explanation

During creation or startup of the queue manager an error occurred while creating or checking a file or directory. Further information detailing the cause of the failure is written to the queue manager error logs.

Response

Identify why the queue manager files cannot be created or why the check failed. It is probable that there is insufficient space on the specified disk, or that there is a problem with access permissions on a file or directory. Correct the problem and submit the command again.

AMQ7020

The operation was carried out, but one or more transactions remain in-doubt.

Severity

10 : Warning

Explanation

The queue manager tried to resolve all internally coordinated transactions which are in-doubt. In-doubt transactions still remain after the queue manager has attempted to deliver the outcome of these transactions to the resource managers concerned. Transactions remain in-doubt when the queue manager cannot deliver the outcome of the transaction to each of the participating resource managers. For example, a resource manager might not be available at this time. Another possibility is that an earlier attempt to resolve the transaction resulted in an unexpected failure, in this case no attempt will be made to resolve the transaction until the queue manager is restarted.

Response

Use the DSPMQTRN command to display the remaining in-doubt transactions.

AMQ7020 (IBM i)

The operation was carried out, but one or more transactions remain in-doubt.

Severity

10 : Warning

Explanation

The queue manager tried to resolve all internally coordinated transactions which are in-doubt. In-doubt transactions still remain after the queue manager has attempted to deliver the outcome of these transactions to the resource managers concerned. Transactions remain in-doubt when the queue manager cannot deliver the outcome of the transaction to each of the participating resource managers. For example, a resource manager might not be available at this time.

Response

Use the Work with Transactions (WRKMQMTRN) command to display the remaining in-doubt transactions.

AMQ7021

An error occurred while deleting the directory structure for the queue manager.

Severity

40 : Stop Error

Explanation

While deleting the queue manager, an error occurred deleting a file or directory. The queue manager might not have been completely deleted.

Response

Follow problem determination procedures to identify the file or directory and to complete deletion of the queue manager.

AMQ7022

The resource manager identification number is not recognized.

Severity

20 : Error

Explanation

The identification number of the resource manager you supplied was not recognized.

Response

Ensure that you entered a valid resource manager identification number. Use the DSPMQTRN command to display a list of resource managers and their identification numbers.

AMQ7023

The resource manager was in an invalid state.

Severity

20 : Error

Explanation

The resource manager, the identification number of which you supplied, was in an invalid state.

Response

Ensure that you entered the correct resource manager identification number. Use the DSPMQTRN command to display a list of resource managers and their identification numbers. A resource manager is in an invalid state, if it is still available to resolve the transaction, use the -a optional flag to resolve this and all other internally coordinated in-doubt transactions.

AMQ7024

Arguments supplied to a command are not valid.

Severity

20 : Error

Explanation

You supplied arguments to a command that it could not interpret. It is probable that you specified a flag not accepted by the command, or that you included extra flags.

Response

Correct the command and submit it again. Additional information on the arguments causing the error might be found in the error logs for the queue, or queue manager, referenced in the command.

AMQ7025

Error in the descriptive text argument (-c parameter) of the crtmqm command.

Severity

40 : Stop Error

Explanation

The descriptive text you supplied to the crtmqm command was in error.

Response

Correct the descriptive text argument and submit the command again.

AMQ7026

A principal or group name was invalid.

Severity

40 : Stop Error

Explanation

You specified the name of a principal or group which does not exist.

Response

Correct the name and resubmit the request.

AMQ7027

Argument <*insert_3*> supplied to command <*insert_4*> is invalid.

Severity

20 : Error

Explanation

The argument <*insert_3*> was supplied to the command <*insert_4*> which could not be interpreted. This argument is either not accepted by the command, or an extra flag has been included.

Response

Correct the command and submit it again.

AMQ7028

The queue manager is not available for use.

Severity

40 : Stop Error

Explanation
You have requested an action that requires the queue manager running, however, the queue manager is not currently running.

Response
Start the required queue manager and submit the command again.

AMQ7030
Quiesce request accepted. The queue manager will stop when all outstanding work is complete.

Severity
0 : Information

Explanation
You have requested that the queue manager end when there is no more work for it. In the meantime, it will refuse new applications that attempt to start, although it allows those already running to complete their work.

Response
None.

AMQ7031
The queue manager is stopping.

Severity
40 : Stop Error

Explanation
You issued a command that requires the queue manager running, however, it is currently in the process of stopping. The command cannot be run.

Response
None

AMQ7041
Object already exists.

Severity
40 : Stop Error

Explanation
A Define Object operation was performed, but the name selected for the object is already in use by an object that is unknown to WebSphere MQ The object name selected by MQ was <*insert_3*>, in directory <*insert_4*>, of object type <*insert_5*>.

Response
Remove the conflicting object from the MQ system, then try the operation again.

AMQ7042
Media image not available for object <*insert_3*> of type <*insert_4*>.

Severity
20 : Error

Explanation
The media image for object <*insert_3*>, type <*insert_4*>, is not available for media recovery. A log file containing part of the media image cannot be accessed.

Response
A previous message indicates which log file could not be accessed. Restore a copy of the log file and all subsequent log files from backup. If this is not possible, you must delete the object instead.

AMQ7042 (IBM i)
Media image not available for object <*insert_3*>.

Severity
20 : Error

Explanation
The media image for object <*insert_3*>, type <*insert_4*>, is not available for media recovery. A log file containing part of the media image cannot be accessed.

Response
A previous message indicates which log file could not be accessed. Restore a copy of the log file and all subsequent log files from backup. If this is not possible, you must delete the object instead.

AMQ7044
Media recovery not allowed.

Severity
20 : Error

Explanation
Media recovery is not possible on a queue manager using a circular log. Damaged objects must be deleted on such a queue manager.

Response
None.

AMQ7047
An unexpected error was encountered by a command.

Severity
40 : Stop Error

Explanation
An internal error occurred during the processing of a command.

Response
Follow problem determination procedures to identify the cause of the error.

AMQ7048
The queue manager name is either not valid or not known

Severity
40 : Stop Error

Explanation
Either the specified queue manager name does not conform to the rules required by WebSphere MQ or the queue manager does not exist. The rules for naming MQ objects are detailed in the WebSphere MQ Command Reference.

Response
Correct the name and submit the command again.

AMQ7048 (Windows)
The queue manager name is either not valid or not known

Severity
40 : Stop Error

Explanation
Either the specified queue manager name does not conform to the rules required by WebSphere MQ or the queue manager does not exist. The rules for naming MQ objects are detailed in the WebSphere MQ Command Reference.

This message can also occur when specifying an option to a command that contains a path. To ensure that the queue manager name is correctly passed to MQ by the Microsoft Windows command interpreter escape all directory separators in the path ("\\") or do not surround the path in quotation marks.

Response
Correct the name and submit the command again.

AMQ7053
The transaction has been committed.

Severity
0 : Information

Explanation
The prepared transaction has been committed.

Response
None.

AMQ7054
The transaction has been backed out.

Severity
0 : Information

Explanation
The prepared transaction has been backed out.

Response
None.

AMQ7055
The transaction number is not recognized.

Severity
20 : Error

Explanation
The number of the transaction you supplied was not recognized as belonging to an in-doubt or heuristically completed transaction.

Response
Ensure that you entered a valid transaction number. It is possible that the transaction number you entered corresponds to a transaction which was committed or backed out before you issued the command to resolve it. It is also possible that the transaction number you entered corresponds to a transaction which is not in the appropriate state for the options you specified. For example, you cannot commit or back out a transaction which is already heuristically completed.

AMQ7056
Transaction number <*insert_1*>,<*insert_2*> is in-doubt.

Severity
0 : Information

Explanation
This message is used to report the number of an in-doubt transaction.

Response
None.

AMQ7059
An error has occurred reading an INI file.

Severity
20 : Error

Explanation
An error has occurred when reading the MQS.INI file or a queue manager QM.INI file.

Response
If you have been changing the INI file content check and correct the change. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7059 (Tandem)
An error has occurred reading an INI file.

Severity
20 : Error

Explanation
An error has occurred when reading the MQSINI file or a queue manager QMINI file.

Response
If you have been changing the INI file content check and correct the change. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7059 (Windows)
An error occurred when reading the configuration data.

Severity
20 : Error

Explanation
An error has occurred when reading the configuration data.

Response
If you have changed the configuration data, check and correct the change. If you have not changed the configuration data, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7060
An error has occurred locking an INI file.

Severity
20 : Error

Explanation
An error has occurred locking the MQS.INI file or a queue manager QM.INI file.

Response
If you have been changing the INI file permissions check and correct the change. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem

identifier and to save any generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7060 (Tandem)
An error has occurred locking an INI file.

Severity
20 : Error

Explanation
An error has occurred locking the MQSINI file or a queue manager QMINI file.

Response
If you have been changing the INI file permissions check and correct the change. If you have not changed the INI file, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7060 (Windows)
An error has occurred locking the configuration data.

Severity
20 : Error

Explanation
An error has occurred locking the configuration data.

Response
If you have changed the configuration data permissions, check and correct the change. If you have not changed the configuration data, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7061
An expected stanza in an INI file is missing or contains errors.

Severity
20 : Error

Explanation
An expected stanza is missing from the MQS.INI file or a queue manager QM.INI file or the stanza contains errors.

Response
If you have been changing the INI file content check and correct the change.

AMQ7061 (Tandem)
An expected stanza in an INI file is missing or contains errors.

Severity
20 : Error

Explanation
An expected stanza is missing from the MQSINI file or a queue manager QMINI file or the stanza contains errors.

Response
If you have been changing the INI file content check and correct the change.

AMQ7061 (Windows)
An expected stanza in the configuration data is missing or contains errors.

Severity
20 : Error

Explanation
An expected stanza is missing from the configuration data or the stanza contains errors.

Response
If you have changed the configuration data, check and correct the change.

AMQ7062
Unable to access an INI file.

Severity
20 : Error

Explanation
Access to the MQS.INI file or a queue manager QM.INI file is denied.

Response
If you have been changing the INI file permissions check and correct the change.

AMQ7062 (Tandem)
Unable to access an INI file.

Severity
20 : Error

Explanation
Access to the MQSINI file or a queue manager QMINI file is denied.

Response
If you have been changing the INI file permissions check and correct the change.

AMQ7062 (Windows)
Unable to access the configuration data.

Severity
20 : Error

Explanation
Access to the configuration data is denied.

Response
If you have changed the configuration data permissions, check and correct the change.

AMQ7063
An INI file is missing.

Severity
20 : Error

Explanation
The MQS.INI file or a queue manager QM.INI file is missing.

Response
If you have been changing the INI file recover the previous file and retry the operation.

AMQ7063 (Tandem)
An INI file is missing.

Severity
20 : Error

Explanation
The MQSINI file or a queue manager QMINI file is missing.

Response
If you have been changing the INI file recover the previous file and retry the operation.

AMQ7063 (Windows)
Configuration data is missing.

Severity
20 : Error

Explanation
The configuration data for WebSphere MQ is missing.

Response
If you have changed the configuration data, recover the previous configuration data and retry the operation.

AMQ7064
Log path not valid or inaccessible.

Severity
40 : Stop Error

Explanation
The supplied log path could not be used by the queue manager. Possible reasons for this include the path not existing, the queue manager not being able to write to the path, or the path residing on a remote device.

Response
Ensure that the log path exists and that the queue manager has authority to read and write to it. If the queue manager already exists, ensure that the LogPath attribute in the queue manager's configuration file matches the intended log path.

AMQ7064 (IBM i)
Auxiliary storage pool identifier not found.

Explanation
The auxiliary storage pool identifier supplied does not exist on the system and could not be used by the queue manager to create a journal receiver.

Response
Specify *SYSTEM, or the identifier of an existing auxiliary storage pool and try the request again. You can use WRKDSKSTS to check the assignment of disk units to auxiliary storage pools.

AMQ7065
Insufficient space on disk.

Severity
40 : Stop Error

Explanation
The operation cannot be completed due to shortage of disk space.

Response
Either make more disk space available, or reduce the disk requirements of the command you issued.

AMQ7066
There are no matching prepared or heuristically completed transactions.

Severity
10 : Warning

Explanation
There are no prepared transactions to be resolved or heuristically completed transactions which match the parameters given.

Response
None.

AMQ7068
Authority file contains an authority stanza that is not valid.

Severity
40 : Stop Error

Explanation
A syntax error has been found in one of the files containing authorization information for the queue manager.

Response
Correct the contents of the incorrect authorization file by editing it.

AMQ7069
The queue manager was created successfully, but cannot be made the default.

Severity
40 : Stop Error

Explanation
The queue manager was defined to be the default queue manager for the machine when it was created. However, although the queue manager has been created, an error occurred trying to make it the default. There might not be a default queue manager defined for the machine at present.

Response
There is probably a problem with the machine-wide ini file. Verify the existence of the file, its access permissions, and its contents. If its backup file exists, reconcile the contents of the two files and then delete the backup. Finally, either update the machine-wide ini file by hand to specify the desired default queue manager, or delete and re-create the queue manager.

AMQ7069 (Windows)
The queue manager was created successfully, but cannot be made the default.

Severity
40 : Stop Error

Explanation
The queue manager was defined to be the default queue manager for the machine when it was created. However, although the queue manager has been created, an error occurred trying to make it the default. There might not be a default queue manager defined for the machine at present.

Response
There is probably a problem with the configuration data. Update the configuration data to specify the desired default queue manager, or delete and re-create the queue manager.

AMQ7072
Invalid QM.INI file stanza. Refer to the error log for more information.

Severity
40 : Stop Error

Explanation
An invalid QM.INI file stanza was found. Refer to the error log for more information.

Response
Correct the error and then retry the operation.

AMQ7072 (Tandem)
Invalid QMINI file stanza. Refer to the error log for more information.

Severity
40 : Stop Error

Explanation
An invalid QMINI file stanza was found. Refer to the error log for more information.

Response
Correct the error and then retry the operation.

AMQ7072 (Windows)
Stanza not valid. Refer to the error log for more information.

Severity
40 : Stop Error

Explanation
A stanza that is not valid was found. Refer to the error log for more information.

Response
Correct the error and retry the operation.

AMQ7073
Log size not valid.

Severity
40 : Stop Error

Explanation
Either the number of log files or the size of the log files was outside the accepted values.

Response
Make sure that the log parameters you enter lie within the valid range.

AMQ7074
Unknown stanza key <*insert_4*> on line <*insert_1*> of ini file <*insert_3*>.

Severity
10 : Warning

Explanation
Line <*insert_1*> of the configuration file <*insert_3*> contained a stanza called <*insert_3*>. This stanza is not recognized.

Response
Check the contents of the file and retry the operation.

AMQ7074 (Windows)
Unknown stanza key <*insert_4*> at <*insert_3*> in the configuration data.

Severity
10 : Warning

Explanation
Key <*insert_3*> contained a stanza called <*insert_4*>. This stanza is not recognized.

Response
Check the contents of the configuration data and retry the operation.

AMQ7074 (IBM i)
Unknown stanza key.

Severity
10 : Warning

Explanation
Line <*insert_1*> of the configuration file <*insert_3*> contained a stanza key <*insert_4*>. This stanza is not recognized.

Response
Check the contents of the file and retry the operation.

AMQ7075
Unknown attribute in ini file.

Severity
10 : Warning

Explanation
Line <*insert_1*> of the configuration file <*insert_3*> contained an attribute called <*insert_4*> that is not valid. This attribute is not recognized in this context.

Response
Check the contents of the file and retry the operation.

AMQ7075 (Windows)
Unknown attribute <*insert_4*> at <*insert_3*> in the configuration data.

Severity
10 : Warning

Explanation
Key <*insert_3*> in the configuration data contained an attribute called <*insert_4*> that is not valid. This attribute is not recognized in this context.

Response
Check the contents of the configuration data and retry the operation.

AMQ7076
Invalid value for attribute in ini file.

Severity
10 : Warning

Explanation
Line <*insert_1*> of the configuration file <*insert_3*> contained value <*insert_5*> that is not valid for the attribute <*insert_4*>.

Response
Check the contents of the file and retry the operation.

AMQ7076 (Windows)
Value <*insert_5*> not valid for attribute <*insert_4*> at <*insert_3*> in the configuration data.

Severity
10 : Warning

Explanation
Key <*insert_3*> in the configuration data contained value <*insert_5*> that is not valid for the attribute <*insert_4*>.

Response
Check the contents of the configuration data and retry the operation.

AMQ7077
You are not authorized to perform the requested operation.

Severity
40 : Stop Error

Explanation
You tried to issue a command for the queue manager. You are not authorized to perform the command.

Response
Contact your system administrator to perform the command for you. Alternatively, request authority to perform the command from your system administrator.

AMQ7078
You entered an object type that is invalid with a generic profile name.

Severity
40 : Stop Error

Explanation
You entered an object type of *ALL or *MQM and an object name that contains generic characters, this is an invalid combination.

Response
Correct the command and submit it again.

AMQ7080
No objects processed.

Severity
10 : Warning

Explanation
No objects were processed, either because no objects matched the criteria given, or because the objects found did not require processing.

Response
None.

AMQ7081
Object <*insert_3*>, type <*insert_4*> recreated.

Severity
0 : Information

Explanation
The object <*insert_3*>, type <*insert_4*> was re-created from its media image.

Response
None.

AMQ7082
Object <*insert_3*>, type <*insert_4*> is not damaged.

Severity
10 : Warning

Explanation
Object <*insert_3*>, type <*insert_4*> cannot be re-created since it is not damaged.

Response
None

AMQ7083
A resource problem was encountered by a command.

Severity
20 : Error

Explanation
The command failed due to a resource problem. Possible causes include the log being full or the command running out of memory.

Response
Look at the previous messages to diagnose the problem. Rectify the problem and retry the operation.

AMQ7084
Object <*insert_3*>, type <*insert_4*> damaged.

Severity
20 : Error

Explanation
The object <*insert_3*>, type <*insert_4*> was damaged. The object must be deleted or, if the queue manager supports media recovery, re-created from its media image.

Response
Delete the object or re-create it from its media image.

AMQ7085
Object <*insert_3*>, type <*insert_4*> not found.

Severity
20 : Error

Explanation
Object <*insert_3*>, type <*insert_4*> cannot be found.

Response
None.

AMQ7086
Media image for object <*insert_3*>, type <*insert_4*> recorded.

Severity
0 : Information

Explanation
The media image for object <*insert_3*>, type <*insert_4*>, defined in Queue Manager <*insert_5*>, has been recorded.

Response
None.

AMQ7087
Object <*insert_3*>, type <*insert_4*> is a temporary object

Severity
20 : Error

Explanation
Object <*insert_3*>, type <*insert_4*> is a temporary object. Media recovery operations are not permitted on temporary objects.

Response
None.

AMQ7088
Object <*insert_3*>, type <*insert_4*> in use.

Severity
20 : Error

Explanation
Object <*insert_3*>, type <*insert_4*> is in use. Either an application has it open or, if it is a local queue, there are uncommitted messages on it.

Response
Ensure that the object is not opened by any applications, and that there are no uncommitted messages on the object, if it is a local queue. Then, retry the operation.

AMQ7089
Media recovery already in progress.

Severity
20 : Error

Explanation
Another media recovery operation is already in progress. Only one media recovery operation is permitted at a time.

Response
Wait for the existing media recovery operation to complete and retry the operation.

AMQ7090 (Windows)
The queue manager CCSID is not valid.

Severity
40 : Stop Error

Explanation
The CCSID to be used by the QMGR is not valid, because:

1) It is a DBCS CCSID.

2) The CCSID encoding is not ASCII or ASCII related. EBCDIC or UCS2 encodings are not valid on this machine.

3) The CCSID encoding is unknown.

Response
Check the CCSID is valid for the machine on which you are working.

AMQ7090 (IBM i)
The queue manager CCSID is not valid.

Severity
40 : Stop Error

Explanation
The CCSID to be used by the QMGR is not valid for the IBM i platform. The CCSID encoding must be a valid EBCDIC value.

Response
Check that the CCSID that you have entered is a valid EBCDIC value.

AMQ7091
You are performing authorization for the queue manager, but you specified an object name.

Severity
40 : Stop Error

Explanation
Modification of authorizations for a queue manager can be performed only from that queue manager. You must not specify an object name.

Response
Correct the command and submit it again.

AMQ7092
An object name is required but you did not specify one.

Severity
40 : Stop Error

Explanation
The command needs the name of an object, but you did not specify one.

Response
Correct the command and submit it again.

AMQ7093
An object type is required but you did not specify one.

Severity
40 : Stop Error

Explanation
The command needs the type of the object, but you did not specify one.

Response
Correct the command and submit it again.

AMQ7094
You specified an object type that is not valid, or more than one object type.

Severity
40 : Stop Error

Explanation
Either the type of object you specified was not valid, or you specified multiple object types on a command which supports only one.

Response
Correct the command and submit it again.

AMQ7095
An entity name is required but you did not specify one.

Severity
40 : Stop Error

Explanation
The command needs one or more entity names, but you did not specify any. Entities can be principals or groups.

Response
Correct the command and submit it again.

AMQ7096
An authorization specification is required but you did not provide one.

Severity
40 : Stop Error

Explanation
The command sets the authorizations on WebSphere MQ objects. However you did not specify which authorizations are to be set.

Response
Correct the command and submit it again.

AMQ7097
You gave an authorization specification that is not valid.

Severity
40 : Stop Error

Explanation
The authorization specification you provided to the command contained one or more items that could not be interpreted.

Response
Correct the command and submit it again.

AMQ7098
The command accepts only one entity name. You specified more than one.

Severity
40 : Stop Error

Explanation
The command can accept only one principal or group name. You specified more than one.

Response
Correct the command and submit it again.

AMQ7099
Entity <*insert_3*> has the following authorizations for object <*insert_4*>:

Severity
0 : Information

Explanation
Informational message. The list of authorizations follows.

Response
None.

AMQ7100
New functions up to command level <*insert_1*> enabled.

Severity
0 : Information

Explanation
The queue manager's command level has been increased and any new function introduced has been enabled for use.

Response
None.

AMQ7104
Resource manager <*insert_1*> has prepared.

Severity
0 : Information

Explanation
This message reports the state of a resource manager with respect to an in-doubt transaction.

Response
None.

AMQ7105
Resource manager <*insert_1*> has committed.

Severity
0 : Information

Explanation
This message reports the state of a resource manager with respect to an in-doubt transaction.

Response
None.

AMQ7106
Resource manager <*insert_1*> has rolled back.

Severity
0 : Information

Explanation
This message reports the state of a resource manager with respect to an in-doubt transaction.

Response
None.

AMQ7107
Resource manager <*insert_1*> is <*insert_3*>.

Severity
0 : Information

Explanation
This message reports the identification number and name of a resource manager.

Response
None.

AMQ7108
Any in-doubt transactions have been resolved.

Severity
0 : Information

Explanation
All, if there were any, of the internally coordinated transactions which were in-doubt, have now been resolved. This message reports successful completion of the RSVMQTRN command when the -a option is used.

Response
None.

AMQ7108 (IBM i)
Any in-doubt transactions have been resolved.

Severity
0 : Information

Explanation
All, if there were any, of the internally coordinated transactions which were in-doubt, have now been resolved.

Response
None.

AMQ7109
A decision on behalf of the unavailable resource manager has been delivered.

Severity
0 : Information

Explanation
A decision for an internally coordinated transaction which was in-doubt, has now been delivered on behalf of the unavailable resource manager. This message reports successful completion of the RSVMQTRN command when the -r option is used.

Response
None.

AMQ7110
Media image for the syncfile recorded.

Severity
0 : Information

Explanation
The media image for the syncfile has been recorded.

Response
None.

AMQ7111
Resource manager <*insert_1*> has participated.

Severity
0 : Information

Explanation
This message reports the state of a resource manager with respect to an in-doubt transaction.

Response
None.

AMQ7112
Transaction number <*insert_1*>,<*insert_2*> has encountered an error.

Severity
0 : Information

Explanation
This message is used to report the number of an in-doubt transaction which has encountered an error with one or more resource managers.

Response
Refer to the queue manager error log for more information about which resource managers are in error. Ensure that the resource managers that were in error, are working correctly, restart the queue manager. If the problem persists, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7113
The Database Name argument, -rn, is missing from the command crtmqm

Severity
20 : Error

Explanation
The required flag, -rn, was omitted from the command crtmqm

Response
Add the flag and associated database name and submit it again.

AMQ7114
The Database Password argument, -rp, is missing from the command crtmqm

Severity
20 : Error

Explanation
The required flag, -rp, was omitted from the command crtmqm

Response
Add the flag and associated database password and submit it again.

AMQ7115
The Database Type argument, -rt, is missing from the command crtmqm

Severity
20 : Error

Explanation
The required flag, -rt, was omitted from the command crtmqm

Response
Add the flag and associated database type and submit it again

AMQ7116
The Database Type argument, -rt, is greater than 8 characters long

Severity
20 : Error

Explanation
The argument supplied with the flag -rt, is greater than 8 characters long

Response
Reduce the length of the database type argument and submit it again

AMQ7117
The MSD shared library failed to load.

Severity
20 : Error

Explanation
The MSD shared library was either not located or failed to load correctly.

Response
Ensure that the database type is specified correctly when creating a queue manager since this is used to form the name of the shared library to be loaded. Further information on the failure might be found in the FFST logs. Also, ensure that the MSD shared library is installed correctly.

AMQ7118
Transaction number <*insert_1*>,<*insert_2*> is heuristically committed.

Severity
0 : Information

Explanation
This message is used to report the number of a heuristically committed transaction.

Response
None.

AMQ7119
Transaction number <*insert_1*>,<*insert_2*> is heuristically rolled back.

Severity
0 : Information

Explanation
This message is used to report the number of a heuristically rolled-back transaction.

Response
None.

AMQ7120
The Trial Period license for this copy of WebSphere MQ has expired.

Severity
20 : Error

Explanation
This copy of WebSphere MQ was licensed to be used in trial mode for a limited period only. This period has expired.

Response
Install a Production license for this copy of WebSphere MQ

AMQ7121
The trial period for this copy of WebSphere MQ has now expired.

Severity
20 : Error

Explanation
This copy of WebSphere MQ was licensed for a limited period only. This period has now expired.

Response
Install a Production license for this copy of WebSphere MQ

AMQ7122
The Trial Period License Agreement was not accepted.

Severity
10 : Warning

Explanation
When the Trial Period License Agreement is displayed, the user must accept it before this copy of WebSphere MQ can be used.

Response
Submit the command again and accept the agreement.

AMQ7123
There is one day left in the trial period for this copy of WebSphere MQ

Severity
0 : Information

Explanation
This copy of WebSphere MQ is licensed for a limited period only.

Response
None.

AMQ7124
This is the final day of the trial period for this copy of WebSphere MQ

Severity
10 : Warning

Explanation
This copy of WebSphere MQ is licensed for a limited period only.

Response
Install a Production license for this copy of WebSphere MQ

AMQ7125
There are <*insert_1*> days left in the trial period for this copy of WebSphere MQ

Severity
0 : Information

Explanation
This copy of WebSphere MQ is licensed for a limited period only.

Response
None.

AMQ7126
This copy of WebSphere MQ is now running in Production mode.

Severity
0 : Information

Explanation
A Production license has been installed for this copy of WebSphere MQ

Response
None.

AMQ7127

Press Enter when you have read the messages

Severity
0 : Information

Explanation
One or more messages have been displayed. They will disappear when the user presses the Enter key.

Response
Press the Enter key when the messages are no longer required.

AMQ7128
No license installed for this copy of WebSphere MQ

Severity
20 : Error

Explanation
The installation of WebSphere MQ is invalid since no Production, Beta, or Trial Period license has been installed.

Response
Check that the installation steps described in the Quick Beginnings documentation have been followed, and if the problem persists use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ7129
The trial period for this copy of WebSphere MQ has already been started.

Severity
0 : Information

Explanation

This copy of WebSphere MQ is licensed for a limited period only and the trial period has been started previously.

Response

None.

AMQ7130

This copy of WebSphere MQ is running in Production mode.

Severity

0 : Information

Explanation

A Production license has been installed for this copy of WebSphere MQ A beta or trial period cannot be started.

Response

None.

AMQ7131

International License Agreement for Evaluation of Programs

Part 1 - General Terms

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE PROGRAM. IBM WILL LICENSE THE PROGRAM TO YOU ONLY IF YOU FIRST ACCEPT THE TERMS OF THIS AGREEMENT. BY USING THE PROGRAM YOU AGREE TO THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE UNUSED PROGRAM TO IBM.

Severity

0 : Information

Explanation

This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7132

The Program is owned by International Business Machines Corporation or one of its subsidiaries (IBM) or an IBM supplier, and is copyrighted and licensed, not sold.

The term "Program" means the original program and all whole or partial copies of it. A Program consists of machine-readable instructions, its components, data, audio-visual content (such as images, text, recordings, or pictures), and related licensed materials.

Severity

0 : Information

Explanation

This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7133

This Agreement includes Part 1 - General Terms and Part 2 - Country Unique Terms and is the complete agreement regarding the use of this Program, and replaces any prior oral or written communications between you and IBM. The terms of Part 2 might replace or modify those of Part 1.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7134

1. License

Use of the Program

IBM grants you a nonexclusive, nontransferable license to use the Program.

You may 1) use the Program only for internal evaluation, testing or demonstration purposes, on a trial or "try-and-buy" basis and 2) make and install a reasonable number of copies of the Program in support of such use, unless IBM identifies a specific number of copies in the documentation accompanying the Program. The terms of this license apply to each copy you make. You will reproduce the copyright notice and any other legends of ownership on each copy, or partial copy, of the Program.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7135

THE PROGRAM MAY CONTAIN A DISABLING DEVICE THAT WILL PREVENT IT FROM BEING USED UPON EXPIRATION OF THIS LICENSE. YOU WILL NOT TAMPER WITH THIS DISABLING DEVICE OR THE PROGRAM. YOU SHOULD TAKE PRECAUTIONS TO AVOID ANY LOSS OF DATA THAT MIGHT RESULT WHEN THE PROGRAM CAN NO LONGER BE USED.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7136

You will 1) maintain a record of all copies of the Program and 2) ensure that anyone who uses the Program does so only for your authorized use and in compliance with the terms of this Agreement.

You may not 1) use, copy, modify or distribute the Program except as provided in this Agreement; 2) reverse assemble, reverse compile, or otherwise translate the Program except as specifically permitted by law without the possibility of contractual waiver; or 3) sublicense, rent or lease the Program.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7137

This license begins with your first use of the Program and ends 1) as of the duration or date specified in the documentation accompanying the Program or 2) when the Program automatically disables itself. Unless IBM specifies in the documentation accompanying the Program that you may retain the Program (in which case, an additional charge might apply), you will destroy the Program and all copies made of it within ten days of when this license ends.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7138

2. No Warranty

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM MAKES NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY. IBM MAKES NO WARRANTY REGARDING THE CAPABILITY OF THE PROGRAM TO CORRECTLY PROCESS, PROVIDE AND/OR RECEIVE DATE DATA WITHIN AND BETWEEN THE 20TH AND 21ST CENTURIES.

This exclusion also applies to any of IBM's subcontractors, suppliers or program developers (collectively called "Suppliers").

Manufacturers, suppliers, or publishers of non-IBM Programs might provide their own warranties.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7139

3. Limitation of Liability

NEITHER IBM NOR ITS SUPPLIERS ARE LIABLE FOR ANY DIRECT OR INDIRECT DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST SAVINGS, OR ANY INCIDENTAL, SPECIAL, OR OTHER ECONOMIC CONSEQUENTIAL DAMAGES, EVEN IF IBM IS INFORMED OF THEIR POSSIBILITY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO YOU.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7140

4. General

Nothing in this Agreement affects any statutory rights of consumers that cannot be waived or limited by contract.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7141

IBM may terminate your license if you fail to comply with the terms of this Agreement. If IBM does so, you must immediately destroy the Program and all copies you made of it.

You may not export the Program.

Neither you nor IBM will bring a legal action under this Agreement more than two years after the cause of action arose unless otherwise provided by local law without the possibility of contractual waiver or limitation.

Neither you nor IBM is responsible for failure to fulfill any obligations due to causes beyond its control.

There is no additional charge for use of the Program for the duration of this license.

IBM does not provide program services or technical support, unless IBM specifies otherwise.

Severity
0 : Information

Explanation

This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7142

Reply 'yes' to accept the Agreement. Reply 'no' if you do not agree to the terms of the Agreement. Reply 'no' and submit the command again, if you want to read the Agreement again.

Severity

0 : Information

Explanation

The Trial Period License Agreement has been displayed to the user and the user should now accept or reject the Agreement.

Response

Reply 'yes' or 'no' and press 'Enter'.

AMQ7143

Press Enter to continue

Severity

0 : Information

Explanation

Part of the Trial Period License Agreement has been displayed to the user. The user should press the Enter key to indicate that they are ready for the next part of the Agreement to be displayed.

Response

Press the Enter key when ready for the next part of the Agreement to be displayed.

AMQ7144

The laws of the country in which you acquire the Program govern this Agreement, except 1) in Australia, the laws of the State or Territory in which the transaction is performed govern this Agreement; 2) in Albania, Armenia, Belarus, Bosnia/Herzegovina, Bulgaria, Croatia, Czech Republic, Georgia, Hungary, Kazakhstan, Kirghizia, Former Yugoslav Republic of Macedonia (FYROM), Moldova, Poland, Romania, Russia, Slovak Republic, Slovenia, Ukraine, and Federal Republic of Yugoslavia, the laws of Austria govern this Agreement; 3) in the United Kingdom, all disputes relating to this Agreement will be governed by English law and will be submitted to the exclusive jurisdiction of the English courts; 4) in Canada, the laws of the Province of Ontario govern this Agreement; and 5) in the United States and Puerto Rico, and People's Republic of China, the laws of the State of New York govern this Agreement.

Severity

0 : Information

Explanation

This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7145

Part 2 - Country Unique Terms

AUSTRALIA:

No Warranty (Section 2):

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, you might have certain rights under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation.

Limitation of Liability (Section 3):

The following paragraph is added to this Section:

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7146

Where IBM is in breach of a condition or warranty implied by the Trade Practices Act 1974, IBM's liability is limited to the repair or replacement of the goods, or the supply of equivalent goods. Where that condition or warranty relates to right to sell, quiet possession or clear title, or the goods are of a kind ordinarily acquired for personal, domestic or household use or consumption, then none of the limitations in this paragraph apply.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7147

NEW ZEALAND:

No Warranty (Section 2):

The following paragraph is added to this Section:

Although IBM specifies that there are no warranties, you might have certain rights under the Consumer Guarantees Act 1993 or other legislation which cannot be excluded or limited. The Consumer Guarantees Act 1993 will not apply in respect of any goods or services which IBM provides, if you require the goods and services for the purposes of a business as defined in the Act.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7148

Limitation of Liability (Section 3):

The following paragraph is added to this Section:

Where products or services are not acquired for the purposes of a business as defined in the Consumer Guarantees Act 1993, the limitations in this Section are subject to the limitations in that Act.

Severity

0 : Information

Explanation

This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7149

GERMANY: No Warranty (Section 2):

The following paragraphs are added to this Section:

The minimum warranty period for Programs is six months.

In case a Program is delivered without specifications, we will only warrant that the Program information correctly describes the Program and that the Program can be used according to the Program information. You have to check the usability according to the Program information within the "money-back guaranty" period.

Limitation of Liability (Section 3):

The following paragraph is added to this Section:

The limitations and exclusions specified in the Agreement will not apply to damages caused by IBM with fraud or gross negligence, and for express warranty.

Severity

0 : Information

Explanation

This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7150

INDIA:

General (Section 4):

The following replaces the fourth paragraph of this Section:

If no suit or other legal action is brought, within two years after the cause of action arose, in respect of any claim that either party might have against the other, the rights of the concerned party in respect of such claim will be forfeited and the other party will stand released from its obligations in respect of such claim.

Severity

0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7151

IRELAND:

No Warranty (Section 2):

The following paragraph is added to this Section:

Except as expressly provided in these terms and conditions, all statutory conditions, including all warranties implied, but without prejudice to the generality of the foregoing all warranties implied by the Sale of Goods Act 1893 or the Sale of Goods and Supply of Services Act 1980 are hereby excluded.

ITALY:

Limitation of Liability (Section 3):

This section is replaced by the following:

Unless otherwise provided by mandatory law, IBM is not liable for any damages which might arise.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7152

UNITED KINGDOM:

Limitation of Liability (Section 3):

The following paragraph is added to this Section at the end of the first paragraph:

The limitation of liability will not apply to any breach of IBM's obligations implied by Section 12 of the Sales of Goods Act 1979 or Section 2 of the Supply of Goods and Services Act 1982.

Severity
0 : Information

Explanation
This is part of the Trial Period License Agreement which must be accepted before a trial period can be started. A trial period allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7153
A license could not be installed for this copy of WebSphere MQ

Severity
20 : Error

Explanation
A Production, Beta or Trial Period license could not be installed for this copy of WebSphere MQ This is because the 'nodelock' file in the 'qmgrs/@SYSTEM' directory could not be created or updated.

Response
Check the ownership and permissions of the 'qmgrs/@SYSTEM' directory.

AMQ7154
The Production license for this copy of WebSphere MQ has expired.

Severity
20 : Error

Explanation
The production license for this copy of WebSphere MQ has an expiry date. This date has been passed.

Response
Contact your IBM support center.

AMQ7155
License file not found or not valid.

Severity
20 : Error

Explanation
The program requires that the License file is present, available and is a valid license file. You can also get this error if you try and use Advanced Message Security (for example, setmqspl) but do not have the AMS component installed.

Response
Check that the installation steps described in the documentation have been followed, and if the problem persists use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ7156
This copy of WebSphere MQ is already running in Production mode.

Severity
0 : Information

Explanation
A Production license has previously been installed for this copy of WebSphere MQ

Response
None.

AMQ7157
The Production license is not valid for this copy of WebSphere MQ

Severity
20 : Error

Explanation
The license <*insert_3*> has been installed but it is not a valid production license for this copy of WebSphere MQ

Response
Submit the SETMQPRD command again specifying the name of a valid production license.

AMQ7158
The Trial Period license is not valid for this copy of WebSphere MQ

Severity
20 : Error

Explanation
The license <*insert_3*> has been installed but it is not a valid trial period license for this copy of WebSphere MQ

Response
Check that the correct version of the file is available.

AMQ7159
A FASTPATH application has ended unexpectedly.

Severity
10 : Warning

Explanation
A FASTPATH application has ended in a way which did not allow the queue manager to clean up the resources owned by that application. Any resources held by the application can only be released by stopping and restarting the queue manager.

Response
Investigate why the application ended unexpectedly. Avoid ending FASTPATH applications in a way which prevents WebSphere MQ from releasing resources held by the application.

AMQ7160
Queue Manager Object

Severity
0 : Information

AMQ7161
Object catalog

Severity
0 : Information

AMQ7162
The setmqaut command completed successfully.

Severity
0 : Information

AMQ7163 (IBM i)
WebSphere MQ job <*insert_2*> started for <*insert_3*>.

Severity
0 : Information

Explanation
The job's PID is <*insert_2*> the CCSID is <*insert_1*>. The job name is <*insert_4*>.

Response
None

AMQ7164 (IBM i)
WebSphere MQ is waiting for a job to start.

Severity
0 : Information

Explanation

WebSphere MQ has been waiting <*insert_1*> seconds to start job <*insert_3*> for Queue Manager: <*insert_5*>

Response

Check that the job queue that is associated with job description <*insert_4*> is not held and that the appropriate maximum active jobs value in the job queue entry is sufficient to allow the job to start. Check that the subsystem that is associated with the job queue is active and has a sufficient value specified for the maximum number of jobs that can be active at the same time.

AMQ7165

The Beta license for this copy of WebSphere MQ has expired.

Severity

20 : Error

Explanation

This copy of WebSphere MQ was licensed to be used for Beta testing for a limited period only. This period has expired.

Response

Install a Production license for this copy of WebSphere MQ

AMQ7166

The Beta period for this copy of WebSphere MQ has now expired.

Severity

20 : Error

Explanation

This copy of WebSphere MQ was licensed for a limited period only. This period has now expired.

Response

Install a Production license for this copy of WebSphere MQ

AMQ7167

The 'Early Release of Programs License Agreement' was not accepted.

Severity

10 : Warning

Explanation

When the IBM International License Agreement for Early Release of Programs is displayed, the user must accept it before this copy of WebSphere MQ can be used.

Response

Submit the command again and accept the agreement.

AMQ7168

There is one day left in the Beta test period for this copy of WebSphere MQ

Severity

0 : Information

Explanation

This copy of WebSphere MQ is licensed for a limited period only.

Response

None.

AMQ7169

This is the final day of the Beta test period for this copy of WebSphere MQ

Severity

10 : Warning

Explanation
This copy of WebSphere MQ is licensed for a limited period only.

Response
Install a Production license for this copy of WebSphere MQ

AMQ7170 (IBM i)
Option is not valid for this transaction.

Severity
20 : Error

Explanation
The Resolve option is not valid for external transactions. The Commit and Backout options are not valid for internal transactions or heuristically completed transactions. The Forget option is only valid for heuristically completed transactions.

Response
Select a different option for this transaction.

AMQ7171
IBM International License Agreement for Early Release of Programs

Part 1 - General Terms

PLEASE READ THIS AGREEMENT CAREFULLY BEFORE USING THE PROGRAM. IBM WILL LICENSE THE PROGRAM TO YOU ONLY IF YOU FIRST ACCEPT THE TERMS OF THIS AGREEMENT. BY USING THE PROGRAM YOU AGREE TO THESE TERMS. IF YOU DO NOT AGREE TO THE TERMS OF THIS AGREEMENT, PROMPTLY RETURN THE UNUSED PROGRAM TO IBM.

Severity
0 : Information

Explanation
This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7172

The Program is owned by International Business Machines Corporation or one of its subsidiaries (IBM) or an IBM supplier, and is copyrighted and licensed, not sold.

The term "Program" means the original program and all whole or partial copies of it. A Program consists of machine-readable instructions, its components, data, audio-visual content (such as images, text, recordings, or pictures), and related licensed materials.

Severity
0 : Information

Explanation
This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7173

The term "Early Release" means that the Program is not formally released or generally available. The term does not imply that the Program will be formally released or made generally available. IBM does not guarantee that a Program formally released or made generally available will be similar to, or compatible with, Early Release versions.

THIS AGREEMENT INCLUDES PART 1 - GENERAL TERMS AND PART 2 - COUNTRY-UNIQUE TERMS AND IS THE COMPLETE AGREEMENT REGARDING THE USE OF THIS PROGRAM, AND REPLACES ANY PRIOR ORAL OR WRITTEN COMMUNICATIONS BETWEEN YOU AND IBM. THE TERMS OF PART 2 MAY REPLACE OR MODIFY THOSE OF PART 1.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7174

1.License

Use of the Program

IBM grants you a nonexclusive, nontransferable license to use the Program.

You may

1) use the Program only for internal evaluation or testing purposes and

2) make and install a reasonable number of copies of the Program in support of such use, unless IBM identifies a specific number of copies in the documentation accompanying the Program. The terms of this license apply to each copy you make. You will reproduce the copyright notice and any other legends of ownership on each copy, or partial copy, of the Program.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7175

THE PROGRAM MAY CONTAIN A DISABLING DEVICE THAT WILL PREVENT IT FROM BEING USED UPON EXPIRATION OF THIS LICENSE. YOU WILL NOT TAMPER WITH THIS DISABLING DEVICE OR THE PROGRAM. YOU SHOULD TAKE PRECAUTIONS TO AVOID ANY LOSS OF DATA THAT MIGHT RESULT WHEN THE PROGRAM CAN NO LONGER BE USED.

You will

1) maintain a record of all copies of the Program and

2) ensure that anyone who uses the Program does so only for your authorized use and in compliance with the terms of this Agreement.

Severity
0 : Information

Explanation
This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7176

You may not

1) use, copy, modify, or distribute the Program except as provided in this Agreement;

2) reverse assemble, reverse compile, or otherwise translate the Program except as specifically permitted by law without the possibility of contractual waiver; or

3) sublicense, rent, or lease the Program.

Severity
0 : Information

Explanation
This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7177

This license begins with your first use of the Program and ends

1) as of the duration or date specified in the documentation accompanying the Program,

2) when the Program automatically disables itself, or

3) when IBM makes the Program generally available. Unless IBM specifies in the documentation accompanying the Program that you may retain the Program (in which case, an additional charge might apply), you will destroy the Program and all copies made of it within ten days of when this license ends.

Severity
0 : Information

Explanation
This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7178

2.No Warranty

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM MAKES NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, THE WARRANTY OF NON-INFRINGEMENT AND THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.. IBM MAKES NO WARRANTY REGARDING

THE CAPABILITY OF THE PROGRAM TO CORRECTLY PROCESS, PROVIDE AND/OR RECEIVE DATE DATA WITHIN AND BETWEEN THE 20TH AND 21ST CENTURIES.

This exclusion also applies to any of IBM's subcontractors, suppliers or program developers (collectively called "Suppliers").

Manufacturers, suppliers, or publishers of non-IBM Programs might provide their own warranties.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7179

3.Limitation of Liability

NEITHER IBM NOR ITS SUPPLIERS ARE LIABLE FOR ANY DIRECT OR INDIRECT DAMAGES, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST SAVINGS, OR ANY INCIDENTAL, SPECIAL, OR OTHER ECONOMIC CONSEQUENTIAL DAMAGES, EVEN IF IBM IS INFORMED OF THEIR POSSIBILITY. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE EXCLUSION OR LIMITATION MAY NOT APPLY TO YOU.

4.Rights In Data

You hereby assign to IBM all right, title, and interest (including ownership of copyright) in any data, suggestions, and written materials related to your use of the Program you provide to IBM. If IBM requires it, you will sign an appropriate document to assign such rights.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7180

5.General

Nothing in this Agreement affects any statutory rights of consumers that cannot be waived or limited by contract.

IBM may terminate your license if you fail to comply with the terms of this Agreement. If IBM does so, you must immediately destroy the Program and all copies you made of it.

You not export the Program.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7181

Neither you nor IBM will bring a legal action under this Agreement more than two years after the cause of action arose unless otherwise provided by local law without the possibility of contractual waiver or limitation.

Neither you nor IBM is responsible for failure to fulfill any obligations due to causes beyond its control.

There is no additional charge for use of the Program for the duration of this license.

Neither of us will charge the other for rights in data or any work performed as a result of this Agreement.

IBM does not provide program services or technical support, unless IBM specifies otherwise.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7182

The laws of the country in which you acquire the Program govern this Agreement, except

1) in Australia, the laws of the State or Territory in which the transaction is performed govern this Agreement;

2) in Albania, Armenia, Belarus, Bosnia/Herzegovina, Bulgaria, Croatia, Czech Republic, Georgia, Hungary, Kazakhstan, Kirghizia, Former Yugoslav Republic of Macedonia (FYROM), Moldova, Poland, Romania, Russia, Slovak Republic, Slovenia, Ukraine, and Federal Republic of Yugoslavia, the laws of Austria govern this Agreement;

3) in the United Kingdom, all disputes relating to this Agreement will be governed by English Law and will be submitted to the exclusive jurisdiction of the English courts;

4) in Canada, the laws of the Province of Ontario govern this Agreement; and

5) in the United States and Puerto Rico, and People's Republic of China, the laws of the State of New York govern this Agreement.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7183

Part 2 - Country-unique Terms

AUSTRALIA: No Warranty (Section 2): The following paragraph is added to this Section: Although IBM specifies that there are no warranties, you might have certain rights under the Trade Practices Act 1974 or other legislation and are only limited to the extent permitted by the applicable legislation.

Limitation of Liability (Section 3): The following paragraph is added to this Section: Where IBM is in breach of a condition or warranty implied by the Trade Practices Act 1974, IBM's liability is limited to the repair or replacement of the goods, or the supply of equivalent goods. Where that condition or warranty relates to right to sell, quiet possession or clear title, or the goods are of a kind ordinarily acquired for personal, domestic or household use or consumption, then none of the limitations in this paragraph apply.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7184

GERMANY: No Warranty (Section 2): The following paragraphs are added to this Section: The minimum warranty period for Programs is six months. In case a Program is delivered without Specifications, IBM will only warrant that the Program information correctly describes the Program and that the Program can be used according to the Program information. You have to check the usability according to the Program information within the "money-back guaranty" period.

Limitation of Liability (Section 3): The following paragraph is added to this Section: The limitations and exclusions specified in the Agreement will not apply to damages caused by IBM with fraud or gross negligence, and for express warranty.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7185

INDIA: General (Section 5): The following replaces the fourth paragraph of this Section: If no suit or other legal action is brought, within two years after the cause of action arose, in respect of any claim that either party might have against the other, the rights of the concerned party in respect of such claim will be forfeited and the other party will stand released from its obligations in respect of such claim.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7186

IRELAND: No Warranty (Section 2): The following paragraph is added to this Section: Except as expressly provided in these terms and conditions, all statutory conditions, including all warranties implied, but without prejudice to the generality of the foregoing, all warranties implied by the Sale of Goods Act 1893 or the Sale of Goods and Supply of Services Act 1980 are hereby excluded.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7187

ITALY: Limitation of Liability (Section 3): This Section is replaced by the following: Unless otherwise provided by mandatory law, IBM is not liable for any damages which might arise.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7188

JAPAN: Rights In Data (Section 4): The following paragraph is added to this Section: You also agree to assign to IBM the rights regarding derivative works, as defined in Articles 27 and 28 of the Japanese Copyright Law. You also agree not to exercise your moral rights.

Severity

0 : Information

Explanation

This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response

None.

AMQ7189

NEW ZEALAND: No Warranty (Section 2): The following paragraph is added to this Section: Although IBM specifies that there are no warranties, you might have certain rights under the Consumer Guarantees Act 1993 or other legislation which cannot be excluded or limited. The

Consumer Guarantees Act 1993 will not apply in respect of any goods or services which IBM provides, if you require the goods and services for the purposes of a business as defined in that Act.

Limitation of Liability (Section 3): The following paragraph is added to this Section: Where Programs are not acquired for the purposes of a business as defined in the Consumer Guarantees Act 1993, the limitations in this Section are subject to the limitations in that Act.

Severity
0 : Information

Explanation
This is part of the Early Release of Programs License Agreement which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7190

UNITED KINGDOM: Limitation of Liability (Section 3): The following paragraph is added to this Section at the end of the first paragraph: The limitation of liability will not apply to any breach of IBM's obligations implied by Section 12 of the Sale of Goods Act 1979 or Section 2 of the Supply of Goods and Services Act 1982.

Severity
0 : Information

Explanation
This is part of the Early Release of Programs License Agreement (VZ125-5544-01 10/97 (MK002))which must be accepted before a Beta test period can be started. A Beta test version allows a copy of WebSphere MQ to be used for a limited period only.

Response
None.

AMQ7191
There are <*insert_1*> days left in the beta test period for this copy of WebSphere MQ

Severity
0 : Information

Explanation
This copy of WebSphere MQ is licensed for a limited period only.

Response
None.

AMQ7192
The Beta test period for this copy of WebSphere MQ has already been started.

Severity
0 : Information

Explanation
This copy of WebSphere MQ is licensed for a limited period only and the Beta test period has been started previously.

Response
None.

AMQ7193
Reply 'yes' to accept the Agreement. Reply 'no' if you do not agree to the terms of the Agreement. Reply 'no' and submit the command again, if you want to read the Agreement again.

Severity
0 : Information

Explanation
The IBM International License Agreement for Early Release of Programs has been displayed to the user and the user should now accept or reject the Agreement.

Response
Reply 'yes' or 'no' and press 'Enter'.

AMQ7194

Press Enter to continue

Severity
0 : Information

Explanation
Part of the IBM International License Agreement for Early Release of Programs has been displayed to the user. The user should press the Enter key to indicate that they are ready for the next part of the Agreement to be displayed.

Response
Press the Enter key when ready for the next part of the Agreement to be displayed.

AMQ7195
The Beta test license is not valid for this copy of WebSphere MQ

Severity
20 : Error

Explanation
The license <*insert_3*> has been installed but it is not a valid trial period license for this copy of WebSphere MQ

Response
Check that the correct version of the file is available.

AMQ7196
By installing this product, you accept the terms of the International Program License Agreement and the License Information supplied with the product.

Severity
0 : Information

Response
None.

AMQ7197
A production or trial license could not be installed for this copy of WebSphere MQ

Severity
20 : Error

Explanation
This copy of WebSphere MQ is a beta version and cannot be used with a production or trial license.

Response
Uninstall the beta version of WebSphere MQ and install the production or trial version.

AMQ7198
Insufficient license units.

Severity
10 : Warning

Explanation
The purchased processor allowance (<*insert_1*>) is less than the number of processors (<*insert_2*>) in this machine.

Response
Ensure sufficient license units have been purchased and use the MQ setmqcap command to set the purchased processor allowance for this installation. Refer to the Quick Beginnings documentation for more information.

AMQ7198 (IBM i)
Insufficient license units.

Severity
10 : Warning

Explanation
The purchased processor allowance for this installation is zero.

Response
Ensure sufficient license units have been purchased and use the MQ CHGMQMCAP command to set the purchased processor allowance for this installation. Refer to the Quick Beginnings documentation for more information.

AMQ7199
The purchased processor allowance is set to <*insert_1*>.

Severity
0 : Information

Explanation
The purchased processor allowance for this installation has been set to <*insert_1*> using the MQ setmqcap command.

Response
None.

AMQ7199 (IBM i)
The purchased processor allowance is set to <*insert_1*>.

Severity
0 : Information

Explanation
The purchased processor allowance for this installation has been set to <*insert_1*> using the MQ CHGMQMCAP command.

Response
None.

AMQ7200
The purchased processor allowance is <*insert_1*>

Severity
0 : Information

Explanation
The purchased processor allowance is currently set to <*insert_1*>

Response
Ensure sufficient license units have been purchased and, if necessary, use the MQ setmqcap command to change the purchased processor allowance for this installation. Refer to the Quick Beginnings documentation for more information.

AMQ7200 (IBM i)
The purchased processor allowance is <*insert_1*>

Severity
0 : Information

Explanation
The purchased processor allowance is currently set to <*insert_1*>

Response
Ensure sufficient license units have been purchased and, if necessary, use the MQ CHGMQMCAP command to change the purchased processor allowance for this installation. Refer to the Quick Beginnings documentation for more information.

AMQ7201
The number of processors in this machine is <*insert_1*>

Severity
0 : Information

Explanation
The operating system reports that the number of processors in this machine is <*insert_1*>

Response
None.

AMQ7202
The number of license units is sufficient for all future possible upgrades to this machine.

Severity
0 : Information

Explanation
The purchased processor allowance for this installation has been set to -1, which allows any permitted processor configuration.

Response
None.

AMQ7203
Purchased processor allowance not set (use setmqcap).

Severity
10 : Warning

Explanation
The purchased processor allowance for this installation has not been set.

Response
Ensure sufficient license units have been purchased and use the MQ setmqcap command to set the purchased processor allowance for this installation. Refer to the Quick Beginnings documentation for more information.

AMQ7203 (IBM i)
Purchased processor allowance not set (use CHGMQMCAP).

Severity
10 : Warning

Explanation
The purchased processor allowance for this installation has not been set.

Response
Ensure sufficient license units have been purchased and use the MQ CHGMQMCAP command to set the purchased processor allowance for this installation. Refer to the Quick Beginnings documentation for more information.

AMQ7203 (IBM i)
Purchased processor allowance not set (use CHGMQMCAP).

Severity
10 : Warning

Explanation
The purchased processor allowance for this installation has not been set.

Response
Ensure sufficient license units have been purchased and use the MQ CHGMQMCAP command to set the purchased processor allowance for this installation. Refer to the Quick Beginnings documentation for more information.

AMQ7204
WebSphere MQ queue manager <*insert_3*> cannot be started by this installation. It has previously been started by a newer release of WebSphere MQ.

Severity
20 : Error

Explanation
The queue manager has previously been started by a newer release of WebSphere MQ at command level <*insert_1*>. This installation is not compatible with the newer release's data. Migration between these releases is not possible.

Response
If the queue manager's data is shared using networked storage, ensure that all installations used to start the queue manager are of the same release. The queue manager can be started by installing a release of WebSphere MQ which supports command level <*insert_1*> or higher.

AMQ7205
WebSphere MQ queue manager <*insert_3*> cannot be started because the authorization service is incompatible with the setting for ClusterQueueAccessControl.

Severity
20 : Error

Explanation
The queue manager has an authorization service at version <*insert_1*> and the queue manager is configured to use ClusterQueueAccessControl=RQMName. The authorization service version is incompatible with this setting for ClusterQueueAccessControl, and so the queue manager cannot be started.

Response
Update the setting for ClusterQueueAccessControl to be XmitQ instead of RQMName, or upgrade the authorization service to a minimum of version MQZAS_VERSION_6.

AMQ7206
Group name has been truncated.

Severity
40 : Stop Error

Explanation
WebSphere MQ only supports group names up to 12 characters long. The operating system is attempting to return a group longer than this.

Response
Reduce the group name to 12 characters or less.

AMQ7207 (Windows)
User ID longer than 12 characters.

Severity
40 : Stop Error

Explanation
WebSphere MQ only supports user names up to 12 characters long. This operation is being attempted from a user name longer than this.

Response
Reduce the user name to 12 characters or less.

AMQ7208
The queue manager failed to pass a PCF message to another queue manager.

Severity
10 : Warning

Explanation
The queue manager attempted to put a PCF message to <*insert_3*> to start the channel <*insert_4*> to cluster queue manager <*insert_5*>. The put failed with reason <*insert_1*>. When the queue manager resolves a cluster queue to a remote cluster queue manager, the message is put to the SYSTEM.CLUS.TRANSMIT.QUEUE. If the channel to the remote cluster queue manager is not running, the queue manager attempts to start the channel by sending a PCF message to <*insert_3*>.

Response
Resolve the problem with <*insert_3*> and if necessary start the channel manually.

AMQ7209
The queue manager attempted to open SYSTEM.CHANNEL.INITQ which failed with reason <*insert_3*>

Severity
10 : Warning

Explanation
When the queue manager resolves a cluster queue to a remote cluster queue manager, the message is put to the SYSTEM.CLUS.TRANSMIT.QUEUE. If the channel to the remote cluster queue manager is not running, the queue manager attempts to start the channel by sending a PCF message to the SYSTEM.CHANNEL.INITQ

Response
Resolve the problem with the SYSTEM.CHANNEL.INITQ and if necessary start the channels manually.

AMQ7210
The Cluster Workload exit module could not be loaded.

Severity
10 : Warning

Explanation
The Cluster Workload exit module <*insert_3*> could not be loaded for reason <*insert_4*>.

Response
Correct the problem with the Cluster Workload exit module <*insert_3*>

AMQ7211
The Queue Manager is still waiting for a reply from the Cluster Workload Exit server process.

Severity
10 : Warning

Explanation
The Queue Manager is configured to run the Cluster Workload Exit in SAFE mode. This means

that the Cluster Workload Exit is run by a server process (amqzlw0). The Queue Manager has been waiting <*insert_1*> seconds for this server process to reply to a request to run the Cluster Workload Exit. It is possible that the exit is hung or is looping.

Response
End the Queue Manager, resolve the problem with the Cluster Workload Exit and restart the Queue Manager

AMQ7212
The address of the Cluster exit function could not be found.

Severity
10 : Warning

Explanation
The address of the Cluster exit function <*insert_4*> could not be found in module <*insert_3*> for reason <*insert_1*> <*insert_5*>.

Response
Correct the problem with the Cluster exit function <*insert_4*> in the module <*insert_3*>

AMQ7214
The module for API Exit <*insert_3*> could not be loaded.

Severity
40 : Stop Error

Explanation
The module <*insert_4*> for API Exit <*insert_3*> could not be loaded for reason <*insert_5*>.

Response
Correct the problem with the API Exit module <*insert_3*>.

AMQ7215
The API Exit <*insert_3*> function <*insert_4*> could not be found in the module <*insert_5*>.

Severity
40 : Stop Error

Explanation
The API Exit <*insert_3*> function <*insert_4*> could not be found in the module <*insert_5*>. The internal return code was <*insert_1*>.

Response
Correct the problem with the API Exit <*insert_3*>.

AMQ7215 (IBM i)
Could not find a function in API Exit <*insert_3*>.

Severity
40 : Stop Error

Explanation
The API Exit <*insert_3*> function <*insert_4*> could not be found in the module <*insert_5*>. The internal return code was <*insert_1*>.

Response
Correct the problem with the API Exit <*insert_3*>.

AMQ7216
An API Exit initialization function returned an error.

Severity
10 : Warning

Explanation
The API Exit <*insert_3*> function <*insert_4*> in the module <*insert_5*> returned CompCode <*insert_1*> and ReasonCode <*insert_2*>.

Response
Correct the problem with the API Exit <*insert_3*>

AMQ7217
The response set by the exit is not valid.

Severity
10 : Warning

Explanation
The API Exit <*insert_3*> module <*insert_4*> function <*insert_5*> returned a response code <*insert_1*> that is not valid in the ExitResponse field of the API Exit parameters (MQAXP).

Response
Investigate why the API Exit <*insert_3*> set a response code that is not valid.

AMQ7219
profile: <*insert_3*>

Severity
0 : Information

AMQ7220
object type: <*insert_3*>

Severity
0 : Information

AMQ7221
entity: <*insert_3*>

Severity
0 : Information

AMQ7222
entity type: <*insert_3*>

Severity
0 : Information

AMQ7223
authority: <*insert_3*>

Severity
0 : Information

AMQ7224
profile: <*insert_3*>, object type: <*insert_4*>

Severity
0 : Information

AMQ7225
No matching authority records.

Severity
0 : Information

Explanation
No authority records match the specified parameters.

AMQ7226
The profile name is invalid.

Severity
20 : Error

Explanation
The profile name contains invalid characters, contains an invalid wildcard specification, or is of invalid length.

Response
Correct the profile name and submit it again.

AMQ7227
WebSphere MQ encountered the following network error: <*insert_3*>

Severity
10 : Warning

Explanation
MQ failed to successfully complete a network operation due to the specified error. If the error is encountered on systems that are part of a Windows 2000 domain it can indicate incorrect DNS or WINS configuration.

Response
Ensure that your network is functioning correctly. On the Windows platform check DNS and/or WINS settings to ensure that domain controllers, used for authentication or authorization functions, are accessible.

AMQ7228 (IBM i)
Display MQ Authority Records for <*insert_3*>

Severity
0 : Information

AMQ7229
<*insert_1*> log records accessed on queue manager <*insert_3*> during the log replay phase.

Severity
0 : Information

Explanation
<*insert_1*> log records have been accessed so far on queue manager <*insert_3*> during the log replay phase in order to bring the queue manager back to a previously known state.

Response
None.

AMQ7230
Log replay for queue manager <*insert_3*> complete.

Severity
0 : Information

Explanation
The log replay phase of the queue manager restart process has been completed for queue manager <*insert_3*>.

Response
None.

AMQ7231
<*insert_1*> log records accessed on queue manager <*insert_3*> during the recovery phase.

Severity
0 : Information

Explanation
<*insert_1*> log records have been accessed so far on queue manager <*insert_3*> during the recovery phase of the transactions manager state.

Response
None.

AMQ7232
Transaction manager state recovered for queue manager <*insert_3*>.

Severity
0 : Information

Explanation
The state of transactions at the time the queue manager ended has been recovered for queue manager <*insert_3*>.

Response
None.

AMQ7233
<*insert_1*> out of <*insert_2*> in-flight transactions resolved for queue manager <*insert_3*>.

Severity
0 : Information

Explanation
<*insert_1*> transactions out of <*insert_2*> in-flight at the time queue manager <*insert_3*> ended have been resolved.

Response
None.

AMQ7234
<*insert_1*> messages from queue <*insert_4*> loaded on queue manager <*insert_3*>.

Severity
0 : Information

Explanation
<*insert_1*> messages from queue <*insert_4*> have been loaded on queue manager <*insert_3*>.

This message might be issued during the WebSphere MQ checkpointing. See Using checkpointing to ensure complete recovery for more details.

Response
None.

AMQ7235 (IBM i)
Queue manager library <*insert_3*> already exists.

Severity
40 : Stop Error

Explanation
The library <*insert_3*> already exists.

Response
Specify a library which does not already exist.

AMQ7236
WebSphere MQ queue manager <*insert_3*> activated.

Severity
0 : Information

Explanation
WebSphere MQ queue manager <*insert_3*> has been activated.

Response
None.

AMQ7237
WebSphere MQ queue manager <*insert_3*> is not a backup queue manager.

Severity
10 : Warning

Explanation
WebSphere MQ queue manager <*insert_3*> is not a backup queue manager and so cannot be activated. A possible reason might be that the queue manager is configured for circular logging.

Response
Re-try the command without the '-a' option.

AMQ7238
WebSphere MQ queue manager <*insert_3*> replay completed.

Severity
0 : Information

Explanation
WebSphere MQ queue manager <*insert_3*> replay has completed.

Response
None.

AMQ7249
WebSphere MQ queue manager <*insert_3*> cannot be started for replay.

Severity
20 : Error

Explanation
WebSphere MQ queue manager <*insert_3*> cannot be started for replay. A possible reason might be that the queue manager is configured for circular logging.

Response
Re-try the command without the '-r' option.

AMQ7250
WebSphere MQ queue manager <*insert_3*> has not been activated.

Severity
20 : Error

Explanation
WebSphere MQ queue manager <*insert_3*> cannot be started because it has previously been started for replay but has not been activated.

Response
Activate the queue manager and try starting the queue manager again.

AMQ7253
The command <*insert_3*> requires one of the following arguments: <*insert_4*>.

Severity
20 : Error

Explanation

The command <*insert_3*> required at least one of the following arguments, none of which you supplied: <*insert_4*>.

Response

Check the WebSphere MQ System Administration documentation for details on the usage of the command, correct the command and then retry.

AMQ7254

Incompatible WebSphere MQ queue manager <*insert_3*> has not been allowed to start.

Severity

20 : Error

Explanation

An attempt to start a <*insert_1*>-bit queue manager was made, this was not allowed as previously this was a <*insert_2*>-bit queue manager. Migration between the previous <*insert_2*>-bit version to current <*insert_1*>-bit version is not possible and would result in an unrecoverable corrupted queue manager.

Response

Either delete this queue manager or uninstall the current <*insert_1*>-bit version and reinstall the previous <*insert_2*>-bit version.

AMQ7255

Arguments supplied to a command are incompatible.

Severity

20 : Error

Explanation

You supplied arguments to a command that it could not interpret. It is probable that you specified one or more flags that cannot be used at the same time.

Response

Correct the command and submit it again. Additional information on the arguments causing the error might be found in the error logs for the queue manager referenced in the command.

AMQ7256

Trace directory <*insert_3*> has restricted permissions <*insert_4*>.

Severity

10 : Warning

Explanation

The directory <*insert_3*> on your system has permissions <*insert_4*>. Some programs might attempt to write trace files to this directory, and fail because of these restricted permissions.

Response

If you want all WebSphere MQ programs on the system to be able to write trace, it is possible these permissions will restrict them from doing so. Please review the permissions and reset them to the product default, as appropriate.

AMQ7257 (Windows)

The MQ service for installation <*insert_2*> (<*insert_3*>) must be running.

Severity

40 : Stop Error

Explanation

The command <*insert_1*> requires the MQ service, amqsvc.exe, and process amqpsrvn.exe, which it launches, to be running.

Response

Ensure that the MQ service is running before issuing the command. Start the service in one of the following ways:

- From an administrative command prompt, issue the command: **<insert_3>\bin\strmqsvc.exe**
- From the Computer Management console, select and start the service named 'IBM WebSphere MQ (<*insert_2*>)' from the list of services shown.

AMQ7258

WebSphere MQ queue manager <*insert_3*> running as a standby.

Severity

0 : Information

Explanation

Queue manager <*insert_3*> is running as a standby instance, ready to become the primary instance if the existing primary instance fails.

Response

None.

AMQ7259

WebSphere MQ queue manager <*insert_3*> could not obtain data lock.

Severity

20 : Error

Explanation

Queue manager <*insert_3*> could not be started because it could not obtain a lock on its data in the file-system. The most likely cause is that the queue manager is running on another computer.

Response

None.

AMQ7260

WebSphere MQ queue manager <*insert_3*> is not permitted to become a standby.

Severity

0 : Information

Explanation

WebSphere MQ queue manager <*insert_3*> could not obtain a lock on its data in the file-system. It was not permitted to become a standby instance waiting to obtain the lock.

Response

None.

AMQ7261

The heuristically completed transaction has been forgotten.

Severity

0 : Information

Explanation

The heuristically completed transaction has now been forgotten by the queue manager.

Response

None.

AMQ7262

<*insert_1*> heuristically completed transactions for queue manager <*insert_3*>.

Severity

0 : Information

Explanation

There are <*insert_1*> heuristically completed transactions for queue manager <*insert_3*>. These transactions will remain heuristically completed until the queue manager is instructed to forget them by the transaction manager or the system administrator.

Response

None.

AMQ7263

Directory is not located on a local filesystem (<*insert_5*>).

Severity

10 : Warning

Explanation

Directory <*insert_4*> appears to be located on a <*insert_5*> file system. Although WebSphere MQ allows you to create this directory on a non-local file system it is not recommended. Please refer to the System Administration Guide for further information on configuring WebSphere MQ to use shared networked file systems.

Response

None.

AMQ7264

IPC directory path is too long.

Severity

40 : Stop Error

Explanation

IPC directory <*insert_3*> is too long for this environment. The length of the IPC directory path is <*insert_1*> characters, however the maximum length allowable is only <*insert_2*> characters.

Response

The length of the IPC directory path can be reduced by specifying a shorter IPC directory prefix when creating the queue manager, or, by shortening the queue manager name.

AMQ7265

Extended message selection available.

Severity

0 : Information

Explanation

A connection has been made by an application capable of performing extended selection of messages on behalf of IBM WebSphere MQ, including on the content of the message. Extended message selection is now available to subscriptions.

Response

None.

AMQ7266

Extended message selection not available.

Severity

0 : Information

Explanation

The application that connected in order to perform extended selection of messages has now disconnected. Extended message selection is no longer available to subscriptions.

Response

None.

AMQ7267
IBM WebSphere MQ configuration information added.

Severity
0 : Information

Explanation
IBM WebSphere MQ configuration information has been added successfully.

Response
None.

AMQ7268
IBM WebSphere MQ configuration information removed.

Severity
0 : Information

Explanation
IBM WebSphere MQ configuration information has been removed successfully.

Response
None.

AMQ7269
A standby instance of queue manager <*insert_5*> has been started. The active instance is running elsewhere.

Severity
0 : Information

Explanation
You tried to start the queue manager but it is already running elsewhere. A standby instance of the queue manager started, ready to become the active instance if the existing active instance fails.

Response
None.

AMQ7270
WebSphere MQ queue manager <insert_3> is already running elsewhere. It permits standby instances.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_3*> could not obtain a lock on its data in the file-system when it was starting. The lock is held by the active instance of the queue manager. The active instance of the queue manager was started permitting standby instances.

Response
If you are trying to start multiple instances of a queue manager to make it highly available, you must start all of the instances using **strmqm -x**.

AMQ7271
IBM WebSphere MQ configuration information does not exist.

Severity
20 : Error

Explanation
IBM WebSphere MQ configuration information does not exist.

Response
None.

AMQ7272
IBM WebSphere MQ configuration information already exists.

Severity
20 : Error

Explanation
IBM WebSphere MQ configuration information already exists.

Response
None.

AMQ7273
Configuration attribute <*insert_3*> must be supplied.

Severity
20 : Error

Explanation
IBM WebSphere MQ configuration attribute <*insert_3*> is required for this stanza.

Response
Supply a value for this attribute and reissue the command.

AMQ7274
IBM WebSphere MQ queue manager <*insert_3*> already has the maximum number of standby instances.

Severity
20 : Error

Exaplanation
You tried to start the queue manager but it is already running elsewhere. It is not possible to start another standby instance because the queue manager has already reached the maximum number of standby instances.

Response
None

AMQ7276
IBM WebSphere MQ queue manager cannot switch over.

Severity
20 : Error

Explanation
You cannot switch over the queue manager. This might be because the queue manager does not have a standby instance or the queue manager is ending.

Response
None

AMQ7279
IBM WebSphere MQ queue manager <insert_3> lost ownership of data lock.

Severity
20 : Error

Explanation
The instance of queue manager <insert_3> has lost ownership of a lock on its data in the file-system due to a transient failure. It was not able to reobtain the lock and will stop automatically to prevent the risk of data corruption.

Response

Check that another instance of the queue manager has become active. Restart this instance of the queue manager as a standby instance. If this problem recurs, it may indicate that the file-system is not sufficiently reliable to support file locking by a multi-instance queue manager.

AMQ7280

WebSphere MQ queue manager <insert_3> appears unresponsive.

Severity

20 : Error

Explanation

The queue manager is monitoring itself for responsiveness. It is not responding sufficiently quickly and will automatically stop if it continues to be unresponsive.

Response

None.

AMQ7282

Library name 'insert_3' is not expected value of 'insert_4'.

Severity

20 : Error

Explanation

The supplied queue manager library name of <*insert_3*> does not match the expected value of <*insert_4*> that was used when queue manager <*insert_5*> was previously created or started.

If a backup or multi-instance queue manager is being configured and the queue manager library is deliberately different between systems, this has the consequence that queue manager journals must be configured.

Response

Check that the library name <*insert_3*> is correct for this queue manager instance. If the library name is incorrect, use the RMVMQMINF command to remove the incorrect information and ADDMQMINF to re-enter the correct configuration information.

AMQ7285

The data contained within file <*insert_3*> cannot be processed by command <*insert_4*>.

Severity

20 : Error

Explanation

The file <*insert_3*> was read by the program insert_4 but the contents of the file were found to be incorrect. Possibly this error occurs because the file <*insert_4*> was incorrectly specified as an argument to command <*insert_4*> or possibly the file is corrupt.

Response

Ensure the file <*insert_3*> is of the required format and submit the command again.

AMQ7286

An error occurred while restoring the cluster cache, see the error logs for details

Severity

10 : Warning

Explanation

One or more errors were detected while restoring the cluster cache. This will not prevent the queue manager from starting but the cluster cache held by this queue manager is now incomplete which may result in inconsistencies in cluster resources visible to and owned by this queue manager. See messages in the error logs for details of the error encountered.

Response
Contact your IBM support center to resolve the problem.

AMQ7287
The command level is outside the range of acceptable values. The value must be at least <*insert_3*> and must not exceed <*insert_4*>.

Severity
20 : Error

Explanation
The command level specified lies outside the range of acceptable values for this command's installation.

Response
Reissue the command specifying a command level in the range of acceptable.

AMQ7288
The queue manager's command level is already <*insert_2*>. No new function has been enabled.

Severity
20 : Error

Explanation
The queue manager's command level is already greater than or equal to the value specified.

Response
None.

AMQ7289
The MQ service for installation <*insert_3*> failed to start with error <*insert_1*>.

Severity
40 : Stop Error

Explanation

The attempt to start the MQ service (`amqsvc.exe`) for installation 'insert_3' failed, the error from the operating system was <*insert_1*>.

The formatted message text for error <*insert_1*> is <*insert_4*> (if blank this indicates that no message text was available).

Response

In order for the MQ service to start it must have been configured to run using the Prepare WebSphere MQ Wizard, if this has not already happened the service may be configured with an invalid userid or be in a 'Disabled' state.

Check that the service named 'IBM WebSphere MQ (insert_3)' has been properly configured and is enabled, then re-issue the command.

AMQ7290
The MQ service for installation <*insert_3*> started successfully.

Severity
0 : Information

Explanation
The MQ service for installation <*insert_3*> was successfully started, or already running.

Response
None.

AMQ7291

Severity
40 : Stop Error

Explanation
The attempt to end the MQ service (`amqsvc.exe`) for installation <*insert_3*> failed, the error from the operating system was <*insert_1*>. The formatted message text for error <*insert_1*> is <*insert_4*> (if blank this indicates that no message text was available).

Response
Check that the service named 'IBM WebSphere MQ <*insert_3*>' has been properly configured and is enabled, then re-issue the command.

AMQ7292
The MQ service for installation <*insert_3*> ended successfully.

Severity
0 : Information

Explanation
The MQ service for installation <*insert_3*> was successfully ended, or already stopped.

Response
None.

AMQ7293
Usage: strmqsvc

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ7294
Usage: endmqsvc

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ7295
IBM WebSphere MQ queue manager <*insert_3*> has not been allowed to start due to migration not being supported.

Severity

Explanation
An attempt to start MQ queue manager <*insert_3*> was made. This was not allowed as previously this queue manager was started by an older version of MQ. Migration between these releases is not supported.

Response
If the queue manager data is shared, ensure that this queue manager is being started on the correct operating system. The queue manager can be started by installing a compatible release of IBM WebSphere MQ. See: http://www-01.ibm.com/software/integration/wmq/requirements

AMQ7305
Trigger message could not be put on an initiation queue.

Severity
10 : Warning

Explanation
The attempt to put a trigger message on queue <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>. The message will be put on the dead-letter queue.

Response
Ensure that the initiation queue is available, and operational.

AMQ7306
The dead-letter queue must be a local queue.

Severity
10 : Warning

Explanation
An undelivered message has not been put on the dead-letter queue <*insert_4*> on queue manager <*insert_5*>, because the queue is not a local queue. The message will be discarded.

Response
Inform your system administrator.

AMQ7307
A message could not be put on the dead-letter queue.

Severity
10 : Warning

Explanation
The attempt to put a message on the dead-letter queue <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>. The message will be discarded.

Response
Ensure that the dead-letter queue is available and operational.

AMQ7308
Trigger condition <*insert_1*> was not satisfied.

Severity
0 : Information

Explanation
At least one of the conditions required for generating a trigger message was not satisfied, so a trigger message was not generated. If you were expecting a trigger message, consult the WebSphere MQ Application Programming Guide for a list of the conditions required. (Note that arranging for condition <*insert_1*> to be satisfied might not be sufficient because the conditions are checked in an arbitrary order, and checking stops when the first unsatisfied condition is discovered.)

Response
If a trigger message is required, ensure that all the conditions for generating one are satisfied.

AMQ7310
Report message could not be put on a reply-to queue.

Severity
10 : Warning

Explanation
The attempt to put a report message on queue <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>. The message will be put on the dead-letter queue.

Response
Ensure that the reply-to queue is available and operational.

AMQ7315
Failed to put message to accounting queue. Reason(<*insert_1*>)

Severity
20 : Error

Explanation
The attempt to put a message containing accounting data to the queue <*insert_3*> failed with reason code <*insert_1*>. The message data has been discarded.

This error message will be written only once for attempts to put a message to the queue as part of the same operation which fail for the same reason.

Response
Ensure that the queue <*insert_3*> is available and operational.

AMQ7316
Failed to put message to statistics queue. Reason(<*insert_1*>)

Severity
20 : Error

Explanation
The attempt to put a message containing statistics data to the queue <*insert_3*> failed with reason code <*insert_1*>. The message data has been discarded.

This error message will be written only once for attempts to put a message to the queue as part of the same operation which fail for the same reason.

Response
Ensure that the queue <*insert_3*> is available and operational.

AMQ7320
Failed to access the retained publication queue. Reason(<*insert_1*>)

Severity
20 : Error

Explanation
An attempt to access messages on the system retained publication queue (<*insert_3*>) failed with reason code <*insert_4*> (<*insert_1*>).

Response
Ensure that the queue <*insert_3*> is available and operational.

AMQ7327
Failed to open topic object <*insert_3*> (referenced by <*insert_4*>)

Severity
20 : Error

Explanation
Each entry in <*insert_4*> must have an existing topic object, which has been created before the entry is added to the namelist.

The topic object <*insert_3*> does not exist, and must be created before that stream or subpoint can be used

Response
Ensure that the topic object <*insert_3*> is available. Remove the entry and add it again to the <*insert_4*> namelist to notify the queue manager to check the topic object again.

AMQ7341 (krcI_CLUSSDR_XMITQ_SWITCHED)
The transmission queue for channel <*insert_1*> is <*insert_3*>.

Severity
00 : Information

Explanation
The switch of transmission queue for channel <*insert one*> was required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue.

This message is written because the queue manager completed switching the transmission queue for channel <*insert one*> to queue <*insert three*>'.

During the switch the queue manager moved <*n*> messages from <*insert two*> to <*insert three*>.

Response
No further action required.

AMQ7342 (krcE_CLUSSDR_XMITQ_SWITCH_FAILED)
WebSphere MQ was unable to display an error message `20007342`

Severity
40 : Error

Explanation
IBM WebSphere MQ attempted to display the message associated with return code `X'20007342'`. The return code indicates that there is no message text associated with the message. Associated with the request are inserts <*n*> : <*m*> : <*insertone*> : <*insert two*> : <*insert three*>.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. See IBM WebSphere MQ support web page, or IBM SupportAssistant web page, to find out if a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7343 (krcE_DYNAMIC_Q_NOT_CREATED)
The request to create a dynamic queue named <*insert one*> failed with reason code <*n*>.

Severity
40 : Error

Explanation
A request was made to create a dynamic queue with the name <*insert one*>, based upon the model queue <*insert two*>, but the operation failed with reason code <*n*>. Possible reasons for the failure include:

- A queue of a different type with the same name already exists.
- The model queue could not be accessed.
- Insufficient resources were available to successfully complete the request.

Response
Use the supplied reason code to correct the cause of the failure and reissue the request.

AMQ7345 (krcE_OPEN_OLD_CLUSTER_XMITQ_FAILED)
Unable to open transmission queue <*insert two*> for channel <*insert one*>.

Severity
40 : Error

Explanation
In order to process the switch of transmission queue the original transmission queue <*insert two*>

must be opened in order to move any pending messages to the new transmission queue. The open request failed with reason code <*n*>. The switch of transmission queue for channel <*insert one*> cannot proceed.

Response

In order to allow the switch to progress, use the reason code provided to identify the cause of the failure and resolve the problem. If the issue cannot be resolved, or the original transmission queue has been deleted, use the **runswchl** command with the **-n** parameter to cause the transmission queue for the channel to be switched without attempting to move any messages from the original transmission queue to the new transmission queue.

Remember: When using this option it is the responsibility of the IBM WebSphere MQ administrator to deal with any messages pending on the original transmission queue.

AMQ7346 (krcE_OPEN_NEW_CLUSTER_XMITQ_FAILED)

Unable to open new transmission queue <*insert two*> for channel <*insert one*>.

Severity

40 : Error

Explanation

In order to process the switch of transmission queue the new transmission queue <*insert three*> must be opened in order to receive any pending messages from the old transmission queue. The open request failed with reason code <*n*>. The switch of transmission queue for channel <*insert one*> cannot proceed.

Response

In order to allow the switch to progress use the reason code provided to identify the cause of the failure and resolve the problem.

AMQ7347 (krcE_INTERNAL_MQGET_FAILED)

MQGET from queue <*insert one*> failed with reason code <*n*>.

Severity

40 : Error

Explanation

An internal MQGET request called as part of a queue manager operation failed with Reason Code <*n*>.

Response

This error message is issued in association with further error messages which explain the implications of this failure. Use the Queue Name <*insert one*> and reason code <*n*> provided in this message in conjunction with the messages which follow to resolve the problem.

AMQ7348 (krcE_INTERNAL_MQPUT_FAILED)

MQPUT to queue <*insert one*> failed with reason code <*n*>.

Severity

40 : Error

Explanation

An internal MQPUT request called as part of a queue manager operation failed with reason code <*n*>.

Response

This error message is issued in association with further error messages which explain the implications of this failure. Use the Queue Name <*insert one*> and reason code <*n*> provided in this message in conjunction with the messages which follow to resolve the problem.

AMQ7349 (krcE_INTERNAL_MQCMIT_FAILED)

MQCMIT failed with reason code <*n*>.

Severity
40 : Error

Explanation
An internal MQCMIT request called as part of a queue manager operation failed with reason code <*n*>.

Response
This error message is issued in association with further error messages which explain the implications of this failure. Use the reason code <*n*> provided in this message in conjunction with the messages which follow to resolve the problem.

AMQ7350 (krcI_CLUSSDR_XMITQ_SWITCH_STARTED)
The switch of transmission queue from queue <*insert two*> to queue <*insert three*> for channel <*insert one*> has been started.

Severity
00 : Information

Explanation
The switch of transmission queue for channel <*insert one*> is required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. This message is written when the process of switching the transmission queue is started.

Response
None.

AMQ7351 (krcI_CLUSSDR_XMITQ_SWITCH_MM_STARTED)
The moving of messages for channel <*insert one*> from transmission queue <*insert two*> to transmission queue <*insert three*> has started.

Severity
00 : Information

Explanation
The switch of transmission queue for channel <*insert one*> is required due to a change to the default cluster transmission queue configuration of the queue manager, or to the cluster channel name attribute of a cluster transmission queue. This message is written when the process of moving messages from the old transmission queue <insert two> to the new transmission queue is started.

If the switch operation is executing as part of a cluster sender channel starting, then the channel continues to run and transfer messages while the moving of messages is completed. If the switch operation is executing as part of the **runswchl** command, the **runswchl** command completes once all of the messages have been moved.

Response
None.

AMQ7352 (krcI_CLUSSDR_XMITQ_MM_STATUS)
<*n*> messages have been moved from queue <*insert two*> to queue <*insert three*>.

Severity
00 : Information

Explanation
The switch of transmission queue for channel <*insert one*> requires that messages are moved from the old transmission queue to the new transmission queue. <*m*> messages have been moved from queue <*insert two*> to queue <*insert three*> <*n*> times. The queue has been empty, but a message arrived before the switch could be completed.

Response

None.

AMQ7353 (krcE_SYNCFILE_UPDATE_FAILED)

Unable to update the channel synchronization file during the switch of transmission queue for channel <*insert one*>.

Severity

40 : Error

Explanation

The queue manager was unable to update the channel synchronization file while completing the change of transmission queue from <*insert two*> to <insert three> for channel <*insert one*>. The reason code for the failure was <*n*>.

Response

Use the reason code provided, and any other failure messages to correct the cause of the failure, before using the **rcrmqobj** command to recover the contents of the channel synchronization file. Run the command:

```
rcrmqobj -m QMgrName -t syncfile
```

The command rebuilds the synchronization file for the queue manager; see rcrmqobj .

AMQ7432 (IBM i)

WebSphere MQ journal entry not available for replay.

Severity

40 : Stop Error

Explanation

A journal replay operation was attempted, but the operation required journal entries from journal receivers that are not currently present on the system.

Response

Restore the required journal receivers from backup. Then try the operation again.

AMQ7433 (IBM i)

An Error occurred while performing a journal replay.

Severity

40 : Stop Error

Explanation

WebSphere MQ encountered a problem reading one or more journal entries while performing a journal replay operation.

Response

If you have previously created a journal receiver for a queue manager or are performing a cold restart of a queue manager, delete the QMQMCHKPT file from the queue manager subdirectory in /QIBM/UserData/mqm/qmgrs/ and attempt to restart the queue manager. If the problem persists, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7434 (IBM i)

The MQ commitment control exit program was called incorrectly. Code <*insert_1*>.

Severity

40 : Stop Error

Explanation
The WebSphere MQ commitment control exit program was called with incorrect parameters.

Response
If the program was called by OS/400 as part of a commit or rollback, save the job log, and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ7435 (IBM i)
The MQ commitment control exit program failed. Code <*insert_1*>.

Severity
40 : Stop Error

Explanation
The WebSphere MQ commitment control exit program failed due to an unexpected error.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ7459 (IBM i)
WebSphere MQ journal receiver <*insert_3*> is the oldest in the chain

Severity
0 : Information

Explanation
The oldest journal receiver in the receiver chain is <*insert_3*> in library <*insert_4*>.

Response
None

AMQ7460 (IBM i)
WebSphere MQ startup journal information.

Severity
0 : Information

Explanation
This message is issued periodically by WebSphere MQ to help you identify which journal receivers can be removed from the system because they are no longer required for startup recovery.

Response
None

AMQ7461 (IBM i)
WebSphere MQ object re-created - reapply authorities.

Severity
0 : Information

Explanation
A previously damaged object has been re-created, either automatically, or by explicit use of the re-create Object (RCRMQMOBJ) command. The authorities that applied to this object have not been re-created.

Response

Use the Grant Authority (GRTMQMAUT) command, as appropriate, to re-create the required authorities to this MQ object.

AMQ7462 (IBM i)

WebSphere MQ media recovery journal information.

Severity

0 : Information

Explanation

This message is issued periodically by WebSphere MQ to help you identify which journal receivers can be removed from the system because they are no longer required for media recovery.

Response

None

AMQ7463

The log for queue manager <*insert_3*> is full.

Severity

20 : Error

Explanation

This message is issued when an attempt to write a log record is rejected because the log is full. The queue manager will attempt to resolve the problem.

Response

This situation might be encountered during a period of unusually high message traffic. However, if you persistently fill the log, you might have to consider enlarging the size of the log. You can either increase the number of log files by changing the values in the queue manager configuration file. You will then have to stop and restart the queue manager. Alternatively, if you need to make the log files themselves bigger, you will have to delete and re-create the queue manager.

AMQ7464

The log for queue manager <*insert_3*> is no longer full.

Severity

0 : Information

Explanation

This message is issued when a log was previously full, but an attempt to write a log record has now been accepted. The log full situation has been resolved.

Response

None

AMQ7465

The log for queue manager <*insert_3*> is full.

Severity

20 : Error

Explanation

An attempt to resolve a log full situation has failed. This is due to the presence of a long-running transaction.

Response

Try to ensure that the duration of your transactions is not excessive. Commit or roll back any old transactions to release log space for further log records.

AMQ7466
There is a problem with the size of the logfile.

Severity
10 : Warning

Explanation
The log for queue manager <*insert_3*> is too small to support the current data rate. This message is issued when the monitoring tasks maintaining the log cannot keep up with the current rate of data being written.

Response
The number of primary log files configured should be increased to prevent possible log full situations.

AMQ7467
The oldest log file required to start queue manager <*insert_3*> is <*insert_4*>.

Severity
0 : Information

Explanation
The log file <*insert_4*> contains the oldest log record required to restart the queue manager. Log records older than this might be required for media recovery.

Response
You can move log files older than <*insert_4*> to an archive medium to release space in the log directory. If you move any of the log files required to re-create objects from their media images, you will have to restore them to re-create the objects. An older log file is one with a numerically smaller log number (but allowing for log number wrapping at 9999999).

AMQ7468
The oldest log file required to perform media recovery of queue manager <*insert_3*> is <*insert_4*>.

Severity
0 : Information

Explanation
The log file <*insert_4*> contains the oldest log record required to re-create any of the objects from their media images. Any log files prior to this will not be accessed by media recovery operations.

Response
Use this information together with the information in the most recent AMQ7467 message. Archivable log files are all those older than BOTH <*insert_4*> and the log file mentioned in the AMQ7467 message.

AMQ7469
Transactions rolled back to release log space.

Severity
0 : Information

Explanation
The log space for the queue manager is becoming full. One or more long-running transactions have been rolled back to release log space so that the queue manager can continue to process requests.

Response
Try to ensure that the duration of your transactions is not excessive. Consider increasing the size of the log to allow transactions to last longer before the log starts to become full.

AMQ7472
Object <*insert_3*>, type <*insert_4*> damaged.

Severity
10 : Warning

Explanation
Object <*insert_3*>, type <*insert_4*> has been marked as damaged. This indicates that the queue manager was either unable to access the object in the file system, or that some kind of inconsistency with the data in the object was detected.

Response
If a damaged object is detected, the action performed depends on whether the queue manager supports media recovery and when the damage was detected. If the queue manager does not support media recovery, you must delete the object as no recovery is possible. If the queue manager does support media recovery and the damage is detected during the processing performed when the queue manager is being started, the queue manager will automatically initiate media recovery of the object. If the queue manager supports media recovery and the damage is detected once the queue manager has started, it can be recovered from a media image using the rcrmqobj command or it can be deleted.

AMQ7472 (IBM i)
Object <*insert_3*>, type <*insert_4*> damaged.

Severity
10 : Warning

Explanation
Object <*insert_3*>, type <*insert_4*> has been marked as damaged. This indicates that the queue manager was either unable to access the object in the file system, or that some kind of inconsistency with the data in the object was detected.

Response
If a damaged object is detected, the action performed depends on whether the queue manager supports media recovery and when the damage was detected. If the queue manager does not support media recovery, you must delete the object as no recovery is possible. If the queue manager does support media recovery and the damage is detected during the processing performed when the queue manager is being started, the queue manager will automatically initiate media recovery of the object. If the queue manager supports media recovery and the damage is detected once the queue manager has started, it can be recovered from a media image using the RCRMQMOBJ command or it can be deleted.

AMQ7477 (IBM i)
WebSphere MQ session no longer active.

Severity
10 : Warning

Explanation
The commitment control exit program was called during a commit or rollback operation. The queue manager was stopped while the program was registered. This might have resulted in the rolling back of some uncommitted message operations.

Response
Inform your system administrator that uncommitted message operations might have been rolled back when the queue manager was stopped.

AMQ7484
Failed to put message to logger event queue. Reason(<*insert_2*>)

Severity
0 : Information

Explanation

The attempt to put a logger event message to the queue <*insert_3*> failed with reason code <*insert_2*>. The message data has been discarded.

Response

Ensure that the queue <*insert_3*> is available and operational. Current logger status information can be displayed with the DISPLAY QMSTATUS runmqsc command.

AMQ7485

Transactions rolled forward to release log space.

Severity

0 : Information

Explanation

The log space for the queue manager is becoming full. One or more long-running prepared transactions have been rolled forward to release log space so that the queue manager can continue to process requests. Equivalent log records for the long-running prepared transactions have been created in the active log.

Response

By the time you read this message, the long-running prepared transaction might already have been resolved automatically. If it is not resolved, this message reappears repeatedly whenever the log space for the queue manager is becoming full.

The following steps assume that there is a prepared transaction that is not being resolved automatically. You should investigate what type of transaction it is, and take steps appropriate to the type of transaction.

Use the DSPMQTRN command to check for externally managed in-doubt transactions and the DISPLAY CHS runmqsc command to check for in-doubt channels.

There are several possible responses:

1. If the long-running transaction is owned by external transaction manager software, then the queue manager cannot decide automatically to resolve it (commit it or roll it back). The queue manager remembers its work for this transaction until the external transaction manager software tells the queue manager the outcome (that is, either to commit the transaction or to roll it back). Therefore you must address this issue through your external transaction manager software, either by issuing commands to it, or (if no such commands exist) by restarting it.
2. If the long-running transaction is owned by an in-doubt channel, then investigate its status. If it will not automatically resolve, then consider using the RESOLVE CHANNEL command.
3. If the long-running transaction is owned by the local queue manager on behalf of an application using MQBEGIN, then perhaps the queue manager has lost contact with external resource manager software that participated in the transaction. Investigate and correct the connectivity from the queue manager to the external resource manager software.
4. If none of the other options succeed, consider using the rsvmqtrn command to tell the queue manager to commit or roll back its work done within the in-doubt transaction. Refer to the description of the rsvmqtrn command in the product documentation.

AMQ7486

Transaction *1111.2222* was preventing log space from being released.

Severity

0 : Information

Explanation

A long running transaction was detected. Message AMQ7469 or AMQ7485 has been issued indicating if the transaction was rolled back or rolled forward in the log to allow the log space to be released. The internal transaction identifier is *1111.2222* which can be correlated with '**dspmqtrn -a**' output. The transaction started at <*insert_1*> and first wrote to the queue manager recovery

log at <*insert_2*>. The following transaction context might be useful in identifying the application causing this behaviour: <*insert_3*>. This message can be correlated with the previous AMQ7469 or AMQ7485 message in the queue manager error logs.

Response

Identify the application responsible for the long running unit of work and ensure this application is creating and completing transactions in a timely manner. If the application is working as expected it may be appropriate to increase the size of the queue manager recovery log.

AMQ7487

Application <*insert_1*> was preventing log space from being released.

Severity

0 : Information

Explanation

A long running transaction was detected, this message is intended to help identify the application associated with this long running transaction. Message AMQ7469 or AMQ7485 has been issued indicating if the transaction was rolled back or rolled forward in the log to allow the log space to be released. Message AMQ7486 has been issued identifying the transaction context of the transaction that was rolled back or rolled forwards. The application associated with this transaction was running with *Pid 1111*, *Tid 2222*, under application name <*insert_1*> and with application description <*insert_2*>. The following application context may also be useful in identifying the application causing this behaviour:<*insert_3*>. This message can be correlated with the previous AMQ7486 message in the queue manager error logs.

Response

Identify the application responsible for the long running unit of work and ensure this application is creating and completing transactions in a timely manner. If the application is working as expected it may be appropriate to increase the size of the queue manager recovery log.

AMQ7540

WebSphere MQ program <*insert_3*> attempted to access file or directory (<*insert_4*>), however it does not exist.

Severity

20 : Error

Explanation

<*insert_3*> is not running as the root UserID, so cannot create the nonexistent file or directory (<*insert_4*>).

Response

If you believe there are existing MQ installations on this machine, or you wish to create a new MQ installation entry, rerun the command as UserID root.

AMQ7541

WebSphere MQ program <*insert_3*> attempted to access file or directory (<*insert_4*>), however access is denied.

Severity

20 : Error

Explanation

<*insert_3*> is not running as the root UserID, and does not have access to file or directory (<*insert_4*>).

Response

Either correct the permissions to allow access to (<*insert_4*>), or rerun the command with sufficient authority.

AMQ7542
WebSphere MQ program <*insert_3*> found that file or directory (<*insert_4*>) permissions were not as expected.

Severity
20 : Error

Explanation
<*insert_3*> is not runnig as the root UserID, so cannot correct file or directory (<*insert_4*>) permissions.

Response
Either correct the permissions to (<*insert_4*>), or rerun the command with sufficient authority to correct the permissions.

AMQ7543
WebSphere MQ program <*insert_3*> found that file (<*insert_4*>) was corrupt but has been repaired.

Severity
0 : Information

Explanation
<*insert_3*> found that file (<*insert_4*>) was corrupt and therefore has been repaired.

Response
Whilst <*insert_3*> has repaired (<*insert_4*>), you may wish to check that the output from WebSphere MQ program dspmqinst reflects the state of the WebSphere MQ installations on this machine.

AMQ7544
WebSphere MQ program <*insert_3*> found that configuration data held in (<*insert_4*>) is corrupt.

Severity
20 : Error

Explanation
<*insert_3*> needs to access MQ configuration data held in (<*insert_4*>), however the data has been corrupted.

Response
Contact your IBM support center.

AMQ7545
WebSphere MQ program <*insert_3*> was supplied an invalid installation path.

Severity
20 : Error

Explanation
<*insert_3*> was supplied with installation path (<*insert_4*>), however this matches an entry with a different installation name.

Response
Correct the installation path and rerun the command.

AMQ7546
WebSphere MQ program <*insert_3*> was supplied an invalid installation name.

Severity
20 : Error

Explanation
<*insert_3*> was supplied with installation name (<*insert_4*>), however this matches an entry with a different installation path.

Response
Correct the installation name and rerun the command.

AMQ7547
Entry created successfully.

Severity
0 : Information

Explanation
<*insert_3*> has successfully created the entry.

Response
None.

AMQ7548
Entry deleted successfully.

Severity
0 : Information

Explanation
<*insert_3*> has successfully deleted the entry.

Response
None.

AMQ7549
Entry does not exist.

Severity
20 : Error

Explanation
<*insert_3*> could not find an entry that matched the supplied parameters.

Response
Use the WebSphere MQ program dspmqinst to display all the WebSphere MQ installations on this machine, then rerun the command with valid parameters.

AMQ7550
Entry is still active and has not been deleted.

Severity
20 : Error

Explanation
<*insert_3*> has found that the entry to be deleted is still an active installation and therefore has not been deleted.

Response
Uninstall the installation then rerun the command.

AMQ7551
Entry uninstalled successfully.

Severity
0 : Information

Explanation
<*insert_3*> has successfully uninstalled the entry.

Response
None.

AMQ7552

WebSphere MQ program <*insert_3*> did not complete successfully.

Severity

20 : Error

Explanation

<*insert_3*> found problems with file (<*insert_4*>) and therefore could not successfully complete the command.

Response

Check the WebSphere MQ error logs and check if there are any FFST files for further details.

AMQ7553

WebSphere MQ program <*insert_3*> did not complete successfully.

Severity

20 : Error

Explanation

<*insert_3*> had an unexpected error and therefore could not successfully complete the command.

Response

Check the WebSphere MQ error logs and check if there are any FFST files for further details.

AMQ7554

WebSphere MQ program <*insert_3*> was supplied an invalid installation descriptive text.

Severity

20 : Error

Explanation

<insert_3> was supplied with installation descriptive text (<insert_4>), however this exceeds the maximum length allowed (<*insert_1*>).

Response

Correct the installation descriptive text and rerun the command.

AMQ7555

```
Usage: crtmqinst ((-n InstName | -p InstPath) [-d Text] )&P  -d   Descriptive text.
&N  -n   Installation name.
&N  -p   Installation path.
```

Severity

0

Explanation

This shows the correct usage.

Response

None.

AMQ7556

```
Usage: dltmqinst (-n InstName | -p InstPath)
&P  -n   Installation name.
&N  -p   Installation path.
```

Severity

0

Explanation

This shows the correct usage.

Response

None.

AMQ7557

```
Usage: dspmqinst [-n InstName | -p InstPath]
&P  -n   Installation name.
&N  -p   Installation path.
```

Severity

0

Explanation

This shows the correct usage.

Response

None.

AMQ7558

WebSphere MQ program <*insert_3*> has detected an invalid installation in path (<*insert_4*>). The minimum supported level of MQ for coexistence with another version of MQ is version: <*insert_5*>. This message may be the result of installing MQ onto a machine which already had an old version of MQ installed; or a FixPack may have been removed from the installation in path (<*insert_4*>).

The configuration of this machine is not supported. You should uninstall or upgrade to the minimum supported level, the installation in path (<*insert_4*>); or uninstall any secondary MQ installations.

Severity

40 : Stop Error

Explanation

<*insert_3*> has detected an invalid installation in path (<*insert_4*>). The minimum supported level of MQ for coexistence with another version of MQ is version: <*insert_5*>. This message may be the result of installing MQ onto a machine which already had an old version of MQ installed; or a FixPack may have been removed from the installation in path (<*insert_4*>).

Response

The configuration of this machine is not supported. You should uninstall or upgrade to the minimum supported level, the installation in path (<insert_4>); or uninstall any secondary MQ installations.

AMQ7559

WebSphere MQ program <*insert_3*> has detected an invalid installation.

Severity

40 : Stop Error

Explanation

<insert_3> has detected an invalid installation in path (<*insert_4*>). The minimum supported level of MQ for coexistence with another version of MQ is version: <*insert_5*>. This message may be the result of installing MQ onto a machine which already had an old version of MQ installed; or a FixPack may have been removed from the installation in path (<*insert_4*>).

Response

The configuration of this machine is not supported. You should uninstall or upgrade to the minimum supported level, the installation in path (<*insert_4*>); or uninstall any secondary MQ installations.

AMQ7560

WebSphere MQ program <*insert_3*> failed to get a lock on file (<*insert_4*>).

Severity

20 : Error

Explanation

<*insert_3*> attempted to lock file (<*insert_4*>) to ensure any reading or writing of the file would not result in the file being corrupted.

Response

The file permissions may be incorrect or another process may be preventing <*insert_3*> to obtain the lock. If it is the latter case, the value supplied here for the process identifier (<*insert_1*>) will be a non zero value, in this case rerun the command when that process has ended.

AMQ7561

WebSphere MQ program <*insert_3*> did not complete successfully due to a lack of system resources.

Severity

20 : Error

Explanation

<*insert_3*> could not obtain system resources such as: storage; handles; disk space, and therefore could not successfully complete the command.

Response

Check the WebSphere MQ error logs and check if there are any FFST files for further details. Rerun the command when sufficient system resources are available.

AMQ7562

WebSphere MQ program <*insert_3*> attempted to access MQ configuration data held in (<*insert_4*>), however access is denied.

Severity

20 : Error

Explanation

<*insert_3*> needs to access MQ configuration data held in (<*insert_4*>) but does not have permission to access it.

Response

Either correct the permissions to allow access to (<*insert_4*>), or rerun the command with sufficient authority.

AMQ7563

Entry modified successfully.

Severity

0 : Information

Explanation

<*insert_3*> has successfully modified the entry.

Response

None

AMQ7601

Duplicate XA resource manager is not valid.

Severity

40 : Stop Error

Explanation

Line <*insert_1*> of the configuration file <*insert_3*> contained a duplicate XA resource manager <*insert_5*>. This is not valid for attribute <*insert_4*>. Each XA resource manager must be given a unique name.

Response

Check the contents of the file and retry the operation.

AMQ7601 (Windows)

Duplicate XA resource manager <*insert_5*> not valid for attribute <*insert_4*> at <*insert_3*> in the configuration data.

Severity

40 : Stop Error

Explanation

Key <*insert_3*> in the configuration data contained a duplicate XA resource manager <*insert_5*>. This is not valid for attribute <*insert_4*>. Each XA resource manager must be given a unique name.

Response

Check the contents of the configuration data and retry the operation.

AMQ7602 (IBM i)

The MQ commitment control exit program was called incorrectly.

Severity

40 : Stop Error

Explanation

The WebSphere MQ commitment control exit program was called with incorrect parameters.

Response

If the program was called by OS/400 as part of a commit or rollback, save the job log, and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ7603

WebSphere MQ has been configured with invalid resource manager <*insert_3*>.

Severity

20 : Error

Explanation

The XA switch file <*insert_4*> for resource manager <*insert_3*> indicates that an attempt has been made to configure another queue manager as an external resource manager. This is not allowed so the queue manager will terminate.

Response

Remove the offending XAResourceManager stanza from the qm.ini configuration file and restart the queue manager.

AMQ7603 (Windows)

WebSphere MQ has been configured with resource manager <*insert_3*> that is not valid.

Severity

20 : Error

Explanation

The XA switch file <*insert_4*> for resource manager <*insert_3*> indicates that an attempt has been made to configure another queue manager as an external resource manager. This is not allowed, so the queue manager will terminate.

Response

Remove the offending XAResourceManager stanza from the configuration data and restart the queue manager.

AMQ7604

The XA resource manager <*insert_3*> was not available when called for <*insert_4*>. The queue manager is continuing without this resource manager.

Severity

10 : Warning

Explanation

The XA resource manager <*insert_3*> has indicated that it is not available, by returning XAER_RMERR on an xa_open request or XAER_RMFAIL when called for something else. Normally this indicates that the resource manager has been shut down. In this case the resource manager cannot participate in any new transactions. Any in-flight transactions in which it was involved will be backed out, and any transactions in which it is in-doubt will only be resolved when contact with the resource manager is re-established. A further message will be issued when the queue manager has been able to do this. If the problem occurred on an xa_open request, and the resource manager should be available, then there might be a configuration problem.

Response

Try to establish the reason why the resource manager is unavailable. It might be that an invalid XAOpenString has been defined for the resource manager in the 'qm.ini' configuration file. If this is the case, stop and then restart the queue manager so that any change will be picked up. Alternatively, the queue manager might be reaching a resource constraint with this resource manager. For example, the resource manager might not be able to accommodate all of the queue manager processes being connected at one time, you might need to alter one of its tuning parameters.

AMQ7604 (IBM i)

The XA resource manager was not available when called.

Severity

10 : Warning

Explanation

The XA resource manager <*insert_3*> has indicated that it is not available, by returning XAER_RMERR on an xa_open request or XAER_RMFAIL when called for <*insert_4*>. The queue manager is continuing without this resource manager. Normally this indicates that the resource manager has been shut down. In this case the resource manager cannot participate in any new transactions. Any in-flight transactions in which it was involved will be backed out, and any transactions in which it is in-doubt will only be resolved when contact with the resource manager is re-established. A further message will be issued when the queue manager has been able to do this. If the problem occurred on an xa_open request, and the resource manager should be available, then there might be a configuration problem.

Response

Try to establish the reason why the resource manager is unavailable. It might be that an invalid XAOpenString has been defined for the resource manager in the 'qm.ini' configuration file. If this is the case, stop and then restart the queue manager so that any change will be picked up. Alternatively, the queue manager might be reaching a resource constraint with this resource manager. For example, the resource manager might not be able to accommodate all of the queue manager processes being connected at one time, you might need to alter one of its tuning parameters.

AMQ7605

The XA resource manager <*insert_3*> has returned an unexpected return code <*insert_1*>, when called for <*insert_4*>.

Severity

20 : Error

Explanation

WebSphere MQ received an unexpected return code when calling XA resource manager <*insert_3*> at its <*insert_4*> entry point. This indicates an internal error, either within MQ or the resource manager.

Response

Try to determine the source of the error. A trace of the failure could be used to look at the XA flows between MQ and the resource manager. MQ has allocated an RMId of <*insert_2*> to this resource manager. This will be useful when isolating the flows associated with the resource manager concerned. If the error occurs on an xa_commit or xa_rollback request, the queue manager will not attempt to redeliver the commit or rollback instruction for this transaction, until after the queue manager has been restarted. The transaction indoubt is identified by the following XID of X<*insert_5*>. If you think that the error lies within the queue manager, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard any information describing the problem until after the problem has been resolved.

AMQ7605 (IBM i)

The XA resource manager has returned an unexpected return code.

Severity

20 : Error

Explanation

WebSphere MQ received unexpected return code <*insert_1*> when calling XA resource manager <*insert_3*> at its <*insert_4*> entry point. This indicates an internal error, either within MQ or the resource manager.

Response

Try to determine the source of the error. A trace of the failure could be used to look at the XA flows between MQ and the resource manager. MQ has allocated an RMId of <*insert_2*> to this resource manager. This will be useful when isolating the flows associated with the resource manager concerned. If the error occurs on an xa_commit or xa_rollback request, the queue manager will not attempt to redeliver the commit or rollback instruction for this transaction, until after the queue manager has been restarted. The transaction indoubt is identified by the following XID of X<*insert_5*>. If you think that the error lies within the queue manager, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard any information describing the problem until after the problem has been resolved.

AMQ7606

A transaction has been committed but one or more resource managers have backed out.

Severity

20 : Error

Explanation

WebSphere MQ was processing the commit operation for a transaction involving external resource managers. One or more of these resource managers failed to obey the commit request and instead rolled back their updates. The outcome of the transaction is now mixed and the resources owned by these resource managers might now be out of synchronization. MQ will issue further messages to indicate which resource managers failed to commit their updates.

Response

The transaction with the mixed outcome is identified by the following XID of X<*insert_3*>. The messages which identify the failing resource managers will also contain this same XID. If the transaction has completed it won't be displayed by the dspmqtrn command and all other transaction participants will have committed their updates. If the transaction is displayed by the dspmqtrn command then there are some participants still in prepared state. In order to preserve data integrity you will need to perform recovery steps local to the failing resource managers.

AMQ7607

A transaction has been rolled back but one or more resource managers have committed.

Severity

20 : Error

Explanation

WebSphere MQ was rolling back a transaction involving external resource managers. One or more of these resource managers failed to obey the rollback request and instead committed their updates. The outcome of the transaction is now mixed and the resources owned by these resource managers might now be out of synchronization. MQ will issue further messages to indicate which resource managers failed to roll back their updates.

Response

The transaction with the mixed outcome is identified by the following XID of X<*insert_3*>. The messages which identify the failing resource managers will also contain this same XID. If the transaction has completed it won't be displayed by the dspmqtrn command and all other transaction participants will have rolled back their updates. If the transaction is displayed by the dspmqtrn command then there are some participants still in prepared state. In order to preserve data integrity you will need to perform recovery steps local to the failing resource managers.

AMQ7608

XA resource manager returned a heuristic return code.

Severity

20 : Error

Explanation

This message is associated with an earlier AMQ7606 message reporting a mixed transaction outcome. It identifies one of the resource managers (<*insert_4*>) that failed to commit its updates. The transaction associated with this failure is identified by the following XID of X<*insert_3*>.

Response

Use the return code <*insert_1*> returned by the resource manager to determine the effects of the failure. The return code indicates that the resource manager made a heuristic decision about the outcome of the transaction which disagrees with the commit decision of the queue manager. In order to preserve data integrity you will need to perform recovery steps local to this resource manager.

AMQ7609

XA resource manager returned a heuristic return code.

Severity

20 : Error

Explanation

This message is associated with an earlier AMQ7607 message reporting a mixed transaction outcome. It identifies one of the resource managers (<*insert_4*>) that failed to roll back its updates. The transaction associated with this failure is identified by the following XID of X<*insert_3*>.

Response

Use the return code <*insert_1*> returned by the resource manager to determine the effects of the failure. The return code indicates that the resource manager made a heuristic decision about the outcome of the transaction which disagrees with the rollback decision of the queue manager. In order to preserve data integrity you will need to perform recovery steps local to this resource manager.

AMQ7612

Switch call exception

Severity
20 : Error

Explanation
Exception number <*insert_1*> occurred when calling resource manager switch <*insert_3*>.

Response
Check the resource manager switch has not been corrupted.

AMQ7622
WebSphere MQ could not load the XA switch load file for resource manager <*insert_3*>.

Severity
20 : Error

Explanation
An error has occurred loading XA switch file <*insert_4*>. If the error occurred during startup then the queue manager will terminate. At all other times the queue manager will continue without this resource manager meaning that it will no longer be able to participate in global transactions. The queue manager will also retry the load of the switch file at regular intervals so that the resource manager will be able to participate again should the load problem be resolved.

Response
Look for a previous message outlining the reason for the load failure. Message AMQ6175 is issued if the load failed because of a system error. If this is the case then follow the guidance given in message AMQ6175 to resolve the problem. In the absence of prior messages or FFST information related to this problem check that the name of the switch load file is correct and that it is present in a directory from which it can be dynamically loaded by the queue manager. The easiest method of doing this is to define the switch load file as a fully-qualified name. Note that if the queue manager is still running it will need to be restarted in order that any changes made to its configuration data can be picked up.

AMQ7623
WebSphere MQ has not been configured with XA resource manager.

Severity
10 : Warning

Explanation
The queue manager has noticed that XA resource manager <*insert_3*> was removed from the qm.ini file of the queue manager. However, it was logged as being involved in <*insert_1*> transactions that are still in-doubt. The queue manager cannot resolve these transactions. The queue manager is continuing without this resource manager.

Response
First check that the qm.ini configuration file of the queue manager concerned hasn't been mistakenly altered resulting in an 'XAResourceManager' stanza being removed, or the 'Name' of any the resource managers being changed. If the qm.ini file was changed by mistake then you will need to reinstate resource manager <*insert_3*> in the qm.ini file before stopping and then restarting the queue manager in order that the change will be picked up. If you have intentionally removed a resource manager from the qm.ini file, consider the integrity implications of your action since the resource manager concerned might be in an in-doubt state. If you are sure that is not the case then you can use the 'rsvmqtrn' command to deliver an outcome on behalf of the resource manager in order that the queue manager can forget about the transactions concerned. If you cannot be sure that such an action will not cause an integrity problem then you should consider re-instating the resource manager in the qm.ini file so that the queue manager can contact the resource manager and automatically resolve the transactions concerned next time the queue manager is restarted.

AMQ7623 (Windows)

WebSphere MQ has not been configured with XA resource manager <*insert_3*> which might be involved in in-doubt transactions. The queue manager is continuing without this resource manager.

Severity

10 : Warning

Explanation

The queue manager has recognized that XA resource manager <*insert_3*> was removed from the registry entry of the queue manager. However, it was logged as being involved in <*insert_1*> transactions that are still in-doubt. The queue manager cannot resolve these transactions.

Response

Check that the configuration data entry of the queue manager concerned has not been altered by mistake, resulting in an 'XAResourceManager' stanza being removed, or the 'Name' of any the resource managers being changed.

If the configuration data entry was changed by mistake, you need to reinstate resource manager <*insert_3*> in the configuration data before stopping, and then restarting the queue manager to access the change.

If you have intentionally removed a resource manager from the configuration data, consider the integrity implications of your action because the resource manager concerned might be in an in-doubt state.

If you are sure that this is not the case, you can use the 'rsvmqtrn' command to instruct the resource manager to inform the queue manager that it can forget about the transactions concerned.

If using the 'rsvmqtrn' command could result in an integrity problem, you should consider reinstating the resource manager in the configuration data, so that the queue manager can contact the resource manager and automatically resolve the transactions concerned next time the queue manager is restarted.

AMQ7624

An exception occurred during an <*insert_4*> call to XA resource manager <*insert_3*>.

Severity

20 : Error

Explanation

An exception has been detected during a call to an XA resource manager. The queue manager will continue after assuming a return code of XAER_RMERR from the call.

Response

An FFST should have been produced which documents the exception. Use this and any further FFSTs to try and determine the reason for the failure. A trace of the problem will be useful to identify the XA flows between the queue manager and the resource manager concerned. MQ has allocated an RMId of <*insert_1*> to this resource manager. Use this to isolate the flows concerned. First contact the supplier of the resource manager for problem resolution. If however you think that the problem lies within the queue manager then save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard any information describing the problem until after it has been resolved.

AMQ7625

The XA resource manager <*insert_3*> has become available again.

Severity
0 : Information

Explanation
WebSphere MQ has managed to regain contact with a resource manager that had become unavailable. Any in-doubt transactions involving this resource manager will be resolved. The resource manager will now be able to participate in new transactions.

Response
None.

AMQ7626
XA resource manager initialization failure. Refer to the error log for more information.

Severity
20 : Error

Explanation
The queue manager has failed to initialize one or more of the XA resource managers defined in the qm.ini configuration file.

Response
Correct the error and restart the queue manager.

AMQ7626 (Windows)
XA resource manager initialization failure. Refer to the error log for more information.

Severity
20 : Error

Explanation
The queue manager has failed to initialize one or more of the XA resource managers defined in the configuration data.

Response
Correct the error and restart the queue manager.

AMQ7627
The XA resource manager <*insert_3*> was not available when called for xa_open. The queue manager is continuing without this resource manager.

Severity
10 : Warning

Explanation
The XA resource manager <*insert_3*> has indicated that it is not available, by returning XAER_RMERR on an xa_open request. Normally this indicates that the resource manager has been shut down. In this case the resource manager cannot participate in any new transactions. Any in-flight transactions in which it was involved will be backed out, and any transactions in which it is in-doubt will only be resolved when contact with the resource manager is re-established. A further message will be issued when the queue manager has been able to do this. If the resource manager should be available, then there might be a configuration problem or another possibility is that you are using a 32-bit instance of Db2, this is not supported on this platform, as WebSphere MQ processes are 64-bit and Db2 does not support 64-bit processes with its 32-bit instances.

Response
Try to establish the reason why the resource manager is unavailable. It might be that an invalid XAOpenString has been defined for the resource manager in the 'qm.ini' configuration file. If this is the case, stop and then restart the queue manager so that any change will be picked up. Alternatively, the queue manager might be reaching a resource constraint with this resource

manager. For example, the resource manager might not be able to accommodate all of the queue manager processes being connected at one time, you might need to alter one of its tuning parameters.

AMQ7701
DMPMQLOG command is starting.

Severity
0 : Information

Explanation
You have started the DMPMQLOG command and it is processing your request.

Response
None.

AMQ7702
DMPMQLOG command has finished successfully.

Severity
0 : Information

Explanation
The DMPMQLOG command has finished processing your request and no errors were detected.

Response
None.

AMQ7703
DMPMQLOG command has used option <*insert_3*> with an invalid value <*insert_4*>.

Severity
20 : Error

Explanation
You started the DMPMQLOG command specifying an invalid option value. The <*insert_4*> value for option <*insert_3*> is either missing or of an incorrect format.

Response
Refer to the command syntax, and then try the command again.

AMQ7704
DMPMQLOG command has used an invalid option <*insert_3*>.

Severity
20 : Error

Explanation
You started the DMPMQLOG command specifying an invalid option of <*insert_3*>.

Response
Refer to the command syntax and then try the command again.

AMQ7705
Usage: dmpmqlog [-b | -s StartLSN | -n ExtentNumber] [-e EndLSN] [-f LogFilePath] [-m QMgrName]

Severity
0 : Information

Response
None.

AMQ7706
DMPMQLOG command has used an incorrect queue manager name <*insert_3*> or path <*insert_4*>.

Severity
20 : Error

Explanation
The DMPMQLOG command has used <*insert_3*> as the queue manager name and, if shown, <*insert_4*> as the directory path for <*insert_3*>. Either <*insert_3*> and/or <*insert_4*> is incorrect; if <*insert_4*> is not shown then it is <*insert_3*> which is incorrect.

Possible reasons for the error include:

that <*insert_3*> is not an existing queue manager name;

the entries for <*insert_3*> in the MQ system initialization (INI) file are incorrect;

<*insert_4*> is not a correct path for <*insert_3*>.

If you started the command specifying option -m (queue manager name option) with a value then this value will have been used as the queue manager name, otherwise the default queue manager name will have been used.

Response
Check that <*insert_3*> is an existing queue manager name. Check your MQ system's initialization (INI) file to ensure that <*insert_3*> and its associated entries are correct. If <*insert_4*> is shown, check that it is a correct MQ system directory path for <*insert_3*>.

AMQ7706 (Windows)
DMPMQLOG command has used an incorrect queue manager name <*insert_3*> or path <*insert_4*>.

Severity
20 : Error

Explanation
The DMPMQLOG command has used <*insert_3*> as the queue manager name and, if shown, <*insert_4*> as the directory path for <*insert_3*>. Either <*insert_3*> and/or <*insert_4*> is incorrect; if <*insert_4*> is not shown then it is <*insert_3*> which is incorrect.

Possible reasons for the error include:

that <*insert_3*> is not an existing queue manager name;

the entries for <*insert_3*> in the MQ configuration data are incorrect;

<*insert_4*> is not a correct path for <*insert_3*>.

If you started the command specifying option -m (queue manager name option) with a value then this value will have been used as the queue manager name, otherwise the default queue manager name will have been used.

Response
Check that <*insert_3*> is an existing queue manager name. Check your MQ configuration data to ensure that <*insert_3*> and its associated entries are correct. If <*insert_4*> is shown, check that it is a correct MQ system directory path for <*insert_3*>.

AMQ7706 (IBM i)
DMPMQLOG command has used an incorrect queue manager name or path.

Severity
20 : Error

Explanation
The DMPMQLOG command has used <*insert_3*> as the queue manager name and, if shown, <*insert_4*> as the directory path for <*insert_3*>. Either <*insert_3*> and/or <*insert_4*> is incorrect; if <*insert_4*> is not shown then it is <*insert_3*> which is incorrect.

Possible reasons for the error include:

that <*insert_3*> is not an existing queue manager name;

the entries for <*insert_3*> in the MQ system initialization (INI) file are incorrect;

<*insert_4*> is not a correct path for <*insert_3*>.

If you started the command specifying option -m (queue manager name option) with a value then this value will have been used as the queue manager name, otherwise the default queue manager name will have been used.

Response
Check that <*insert_3*> is an existing queue manager name. Check your MQ system's initialization (INI) file to ensure that <*insert_3*> and its associated entries are correct. If <*insert_4*> is shown, check that it is a correct MQ system directory path for <*insert_3*>.

AMQ7707
DMPMQLOG command has failed: CompCode = 0x<*insert_1*>.

Severity
20 : Error

Explanation
The DMPMQLOG command has detected an error and the MQ recording routine has been called. Possible reasons for this include a damaged log file, a problem during initialization for the queue manager or an internal MQ failure.

Response
Check that the queue manager being used by DMPMQLOG, as specified by you using the -m command option or defaulted, exists and is not currently running. If it does not exist, try the command again specifying an existing queue manager. If it is running, stop the queue manager and then try the command again. Otherwise, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Note the completion code (CompCode) and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ7708
DMPMQLOG command has used an invalid default queue manager name.

Severity
20 : Error

Explanation
You started the DMPMQLOG command without specifying option -m (queue manager name option) and so your MQ default queue manager name has been used. However, this default name either could not be found or is invalid.

Response
Check that the default queue manager name exists and is valid, and then try the command again.

AMQ7709
DMPMQLOG command has used an invalid combination of options.

Severity
20 : Error

Explanation
You started the DMPMQLOG command specifying an invalid combination of the options -b (base LSN option), -s (start LSN option) and -n (extent number option). Only 1 or none of these options can be specified.

Response
Refer to the command syntax and then try the command again.

AMQ7710
DMPMQLOG command has used option -n which is invalid for circular logging.

Severity
20 : Error

Explanation
You started the DMPMQLOG command specifying option -n (extent number option) but this is not valid when your MQ log is defined as circular.

Response
Use a different option and then try the command again.

AMQ7711
DMPMQLOG command has used option -m with a value that is too long.

Severity
20 : Error

Explanation
You started the DMPMQLOG command specifying option -m (queue manager name option) with a value that is more than <*insert_1*> characters.

Response
Specify a shorter queue manager name and then try the command again.

AMQ7712
DMPMQLOG command has used option -f with a value which is too long.

Severity
20 : Error

Explanation
You started the DMPMQLOG command specifying option -f (log file path option) with a value which is more than <*insert_1*> characters.

Response
Specify a shorter log file path name and then try the command again.

AMQ7713
DMPMQLOG command was unable to allocate sufficient storage.

Severity
20 : Error

Explanation
The DMPMQLOG command has been unable to allocate some storage.

Response
Free some storage and then try the command again.

AMQ7714
DMPMQLOG command has reached the end of the log.

Severity
0 : Information

Explanation
The DMPMQLOG command has processed any log data and has now reached the end of the log.

Response
None.

AMQ7715
DMPMQLOG command cannot open file <*insert_3*>.

Severity
20 : Error

Explanation
The DMPMQLOG command was unable to open file <*insert_3*> for reading.

Response
Check that the file exists, can be opened for reading, and that you have authority to access it, and then try the command again.

AMQ7716
DMPMQLOG command has finished unsuccessfully.

Severity
0 : Information

Explanation
The DMPMQLOG command has finished with your request but an error has been detected. The previous message issued by the command can be used to identify the error.

Response
Refer to the previous message issued by the command.

AMQ7717
DMPMQLOG command has failed to initialize: CompCode = 0x<*insert_1*>.

Severity
20 : Error

Explanation
The DMPMQLOG command has failed during its initialization and the MQ recording routine has been called. Possible reasons for this include that your queue manager is already running. The completion code can be used to identify the error.

Response
Check that the queue manager being used by DMPMQLOG, as specified by you using the -m command option or defaulted, exists and is not currently running. If it is running, stop the queue manager and then try the command again. Otherwise, use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7718
DMPMQLOG command is using a default of <*insert_3*> for the queue manager name.

Severity
0 : Information

Explanation
You have started the DMPMQLOG command without specifying option -m (queue manager name option) and so a default value of <*insert_3*> is being used. This value is obtained from your default queue manager name.

Response
None.

AMQ7718 (IBM i)
DMPMQLOG command is using the default queue manager name.

Severity
0 : Information

Explanation
You have started the DMPMQLOG command without specifying option -m (queue manager name option) and so a default value of <*insert_3*> is being used. This value is obtained from your MQ default queue manager name.

Response
None.

AMQ7719
DMPMQLOG command is using a default of <*insert_3*> for the starting dump location.

Severity
0 : Information

Explanation
You have started the DMPMQLOG command without specifying option -b (base LSN option), option -s (start LSN option) or option -n (extent number option), and so a default value of <*insert_3*> is being used. This value is the Log Sequence Number (LSN) of the first record in the active part of the log, and is used as the location from which to start dumping.

Response
None.

AMQ7719 (IBM i)
DMPMQLOG command is using the default starting dump location.

Severity
0 : Information

Explanation
You have started the DMPMQLOG command without specifying option -b (base LSN option), option -s (start LSN option) or option -n (extent number option), and so a default value of <*insert_3*> is being used. This value is the Log Sequence Number (LSN) of the first record in the active part of the log, and is used as the location from which to start dumping.

Response
None.

AMQ7720
DMPMQLOG command is using extent <*insert_1*> but the current extent is <*insert_2*>.

Severity
20 : Error

Explanation
You have started the DMPMQLOG command specifying option -n (extent number option) with a value of <*insert_1*> but this value is greater than <*insert_2*>, which represents the extent currently being used.

Response
When using option -n, specify its value as being less than or equal to the extent number currently being used.

AMQ7721
DMPMQLOG command has not found any log records in extent number <*insert_1*>.

Severity
0 : Information

Explanation
During its normal processing, the DMPMQLOG command did not find any log records in this extent.

Response
None.

AMQ7722
DMPMQLOG command cannot find the object catalog for queue manager <*insert_3*>.

Severity
20 : Error

Explanation
The DMPMQLOG command is using the queue manager named <*insert_3*> but cannot find the manager's object catalog file. This file should have been created at the time the queue manager was created.

Response
Refer to the "System Management Guide" for a description of the location and name of the object catalog file. Check that the file exists and is available for use by this command. If it does not exist then you will need to re-create the queue manager.

AMQ7722 (IBM i)
DMPMQLOG command cannot find the object catalog for the queue manager.

Severity
20 : Error

Explanation
The DMPMQLOG command is using the queue manager named <*insert_3*> but cannot find the manager's object catalog file. This file should have been created at the time the queue manager was created.

Response
Refer to the "System Management Guide" for a description of the location and name of the object catalog file. Check that the file exists and is available for use by this command. If it does not exist then you will need to re-create the queue manager.

AMQ7723
DMPMQLOG command cannot find the requested Log Sequence Number (LSN).

Severity
20 : Error

Explanation
The DMPMQLOG command has been started with an LSN but it cannot be found in the log.

Response
Check for an existing LSN and then try the command again.

AMQ7724
DMPMQLOG command cannot use the requested extent number.

Severity
20 : Error

Explanation
The DMPMQLOG command has been started with an extent number but it is beyond the end of the log.

Response
Check for an existing extent number and then try the command again.

AMQ7725
DMPMQLOG command cannot find an old Log Sequence Number (LSN).

Severity
20 : Error

Explanation
The DMPMQLOG command has been started specifying an LSN which is older than the log's base LSN. However, the specified LSN could not be found.

Response
Check for an existing LSN and then try the command again.

AMQ7726
DMPMQLOG command has used option -s with an incorrect value for circular logging.

Severity
20 : Error

Explanation
You started the DMPMQLOG command specifying option -s (start LSN option) with a value which is less than the base LSN of a log which is defined as circular. LSN values less than the base LSN can only be specified when using a linear log.

Response
When using option -s with a circular log, specify an option value which is equal or greater to the log's base LSN, and then try the command again.

AMQ7751 (IBM i)
MIGRATEMQM program is starting.

Severity
0 : Information

Explanation
You have started the MIGRATEMQM program.

Response
None.

AMQ7752 (IBM i)
MIGRATEMQM has completed successfully.

Severity
0 : Information

Explanation
The MIGRATEMQM program has completed migration of your queue manager and no errors were detected.

Response
None.

AMQ7753 (IBM i)
MIGRATEMQM has failed due to errors.

Severity
20 : Error

Explanation
See the previously listed messages in the job log. Correct the errors and then restart the MIGRATEMQM program.

Response
None.

AMQ7754 (IBM i)
MIGRATEMQM has detected an error and is unable to continue.

Severity
20 : Error

Explanation
See the previously listed messages in this job log, or in associated job logs. Correct the errors and then restart the MIGRATEMQM program.

Response
None.

AMQ7755 (IBM i)
Unable to locate a required journal receiver.

Severity
20 : Error

Explanation
The MIGRATEMQM program attempted to locate the journal receivers to use for migration, but the operation required access to a journal or journal receiver that is not currently present on the system.

Response
Restore the required journal or journal receiver from backup. Then restart the MIGRATEMQM program.

AMQ7756 (IBM i)
Unable to locate a required journal entry.

Severity
20 : Error

Explanation
The MIGRATEMQM program was unable to retrieve a journal entry required for migration. The operation might have failed because a required journal receiver is not currently present on the system.

Response
Restore the required journal receiver from backup. Then restart the MIGRATEMQM program.

AMQ7757 (IBM i)
Queue manager <*insert_3*> already exists.

Severity
20 : Error

Explanation
The MIGRATEMQM program is unable to create a queue manager with the same name as used in the previous release because a queue manager of this name has already been created.

Response
Delete the queue manager. Then restart the MIGRATEMQM program.

AMQ7758 (IBM i)
Queue manager starting.

Severity
0 : Information

Explanation
The queue manager "<*insert_3*>" is starting.

Response
None.

AMQ7759 (IBM i)
Recreating WebSphere MQ objects.

Severity
0 : Information

Explanation
WebSphere MQ objects are being re-created from their media images contained in the log.

Response
None.

AMQ7760 (IBM i)
Recreating WebSphere MQ channels.

Severity
0 : Information

Explanation
WebSphere MQ channels are being re-created from the previous channel definition file.

Response
None.

AMQ7761 (IBM i)
Unexpected return code from command <*insert_3*>.

Severity
20 : Error

Explanation
An unexpected return code, <*insert_1*>, was returned by command <*insert_3*>.

Response
See the previously listed messages in this job log, or in associated job logs.

AMQ7762 (IBM i)
Unexpected error from channel migration.

Severity
20 : Error

Explanation
The migration of channel definitions or channel synchronization data encountered an unexpected error.

Response
See the previously listed messages in this job log, or in associated job logs.

AMQ7770
Sent file <*insert_3*>

Severity
40 : Stop Error

Explanation
The file was successfully sent.

Response
None.

AMQ7771
Received file.

Severity
40 : Stop Error

Explanation
The file was successfully received.

Response
None.

AMQ7772
Complete file list

Severity
40 : Stop Error

Explanation
Displays a list of complete files.

Response
None.

AMQ7773
Incomplete file list

Severity
40 : Stop Error

Explanation
Displays a list of incomplete files.

Response
None.

AMQ7774
Other message list

Severity
40 : Stop Error

Explanation
Displays a list of other messages.

Response
None.

AMQ7775
Nothing to list.

Severity
40 : Stop Error

Explanation
Nothing to list.

Response
None.

AMQ7776
Deleted.

Severity
40 : Stop Error

Explanation
File deleted.

Response
None.

AMQ7777
Nothing to delete.

Severity
40 : Stop Error

Explanation
Nothing to delete.

Response
None.

AMQ7778
Syntax error. The correct syntax is:

Severity
40 : Stop Error

Explanation
Invalid arguments supplied.

Response
One or more options were incorrectly specified when issuing the send or receive command. Check the options used and reissue the command.

AMQ7779
Cannot connect to default queue manager.

Severity
40 : Stop Error

Explanation
Queue manager not available.

Response
Check that the queue manager exists and that the listener is running.

AMQ7780
Cannot connect to queue manager *<insert_3>*

Severity
40 : Stop Error

Explanation
Queue manager not available.

Response
Check that the queue manager exists and that the listener is running.

AMQ7781
Application memory unavailable.

Severity
40 : Stop Error

Explanation
There is insufficient memory to perform the requested action.

Response

1) Check the message size is not excessive

2) Close other applications and try the command again

AMQ7783
Queue name required.

Severity
40 : Stop Error

Explanation
A queue name was not specified when issuing a send or receive command.

Response
Reissue the command with the QueueName option.

AMQ7784
Cannot open queue <*insert_3*>

Severity
40 : Stop Error

Explanation
Cannot open queue <*insert_3*>

Response
Check that the queue exists.

AMQ7785
Cannot open file <*insert_3*>

Severity
40 : Stop Error

Explanation
Cannot open file <*insert_3*>

Response
Check that the file exists, that it is in the correct location and has the appropriate file permissions.

AMQ7786
Cannot put to queue <*insert_3*>

Severity
40 : Stop Error

Explanation
Cannot put to queue <*insert_3*>

Response

1) Check the Queue Manager has sufficient log space for sending large messages

2) Check the queue does not have put inhibited

3) Check the queue is not full

4) Check the message size of the queue is greater than the message size

5) Check the user has sufficient authority to put messages on the queue

AMQ7787
No file name specified.

Severity
40 : Stop Error

Explanation
No file name specified.

Response

A file name was not specified when issuing a send command. Reissue the command with the FileName option.

AMQ7788

Message length is too small to send data.

Severity

40 : Stop Error

Explanation

Message length is too small to send data.

Response

Increase the message size and resend with a send command, using the -l MessageSize option to specify a larger message size.

AMQ7789

Sending file has changed.

Severity

40 : Stop Error

Explanation

The file being sent has been changed before the complete file has been sent.

Response

Check the file for integrity and reissue the send command.

AMQ7790

Cannot get from queue <*insert_3*>

Severity

40 : Stop Error

Explanation

The list, get, delete or extract request has failed.

Response

1) Check the queue does have get inhibited

2) Check the user has sufficient WebSphere MQ authority to get messages from the queue

AMQ7791

Cannot write to file.

Severity

40 : Stop Error

Explanation

The get or extract request has failed.

Response

1) Check that the file is not write-protected. In Windows Explorer, right-click the file name and select Properties. Check the user has sufficient authority to write to the destination file system.

2) Check the destination file system exists

3) Check the destination file system is not full

AMQ7792

CorrelId is invalid.

Severity

40 : Stop Error

Explanation
CorrelId is invalid.

Response

1) Check that a valid correlation ID has been specified when receiving a file with the -c option.

2) It must be 48 characters in length.

3) Use the -v option of the receive command to display the correlation ID.

AMQ7793
MsgId is invalid.

Severity
40 : Stop Error

Explanation
MsgId is invalid.

Response

1) Check that a valid message ID has been specified when receiving an 'other' message with the -u option.

2) It must be 48 characters in length.

AMQ7794
No messages to receive.

Severity
40 : Stop Error

Explanation
There are no FTA files on the specified queue.

Response
Check with the sender that the file was actually sent.

AMQ7795
Cannot delete the file because it's not unique.

Severity
40 : Stop Error

Explanation
Cannot delete the file because it's not unique.

Response
None.

AMQ7796
Cannot replace an existing file.

Severity
40 : Stop Error

Explanation
Cannot replace an existing file.

Response
Reissue the command with the -y option.

AMQ7797
Unable to load the WebSphere MQ library.

Severity
40 : Stop Error

Explanation
Unable to load the WebSphere MQ library.

Response
None.

AMQ7798
Unable to locate <*insert_3*>.

Severity
40 : Stop Error

Explanation
This application requires <*insert_3*>.

Response
Check that <*insert_3*> is available and installed correctly.

AMQ7799
Unable to start <*insert_3*>.

Severity
40 : Stop Error

Explanation
This application cannot start <*insert_3*>.

Response
Check that <*insert_3*> is available and installed correctly.

AMQ7800
CorrelId <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ7801
Dir <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ7802
UserData <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ7803
FileName <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ7804
Length <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ7805
MsgId <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ7806
Could not start WebSphere MQ web administration server: <*insert_1*>.

Severity
0 : Information

Explanation
An unsuccessful attempt was made to start the web administration server on port <*insert_1*>.

Response
Check the product is installed correctly; the required registry keys and values are correct and the web server port is not already in use. If the problem persists save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ7807
WebSphere MQ web administration server running.

Severity
0 : Information

Explanation
WebSphere MQ web administration server running. Listening on port <*insert_4*>, root directory is <*insert_5*>.

Response
No action is required.

AMQ7808
Internal run-time error in WebSphere MQ web administration: <*insert_4*>.

Severity
0 : Information

Explanation
WebSphere MQ web administration had the following internal run-time error: <*insert_4*>.

Response
Check that: the product is installed correctly and that the required registry keys and values are correct. If the problem persists save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ7809
WebSphere MQ Publish/Subscribe web administration user limit reached.

Severity
10 : Warning

Explanation
The maximum number of concurrent web administration users has been reached (<*insert_4*>).

Response
Use the 'Web Administration Server' properties page in the Microsoft Management Console to increase the value of the web administration 'MaxClients' parameter.

AMQ7810 (Windows)
Failed to create class, reason code: <*insert_1*>.

Severity
20 : Error

Explanation
While trying to create class <*insert_3*> on <*insert_4*> error code <*insert_1*> was encountered. The associated error message generated by the operating system is: <*insert_5*>

Response
Check the system documentation to determine the course of action required to rectify the problem.

AMQ7880 (Windows)
Error code <*insert_1*> starting <*insert_4*>/<*insert_3*> WebSphere MQ service.

Severity
0 : Information

Explanation
The service was unable to start <*insert_4*>/<*insert_3*>. The error message reported was as follows: <*insert_5*>

Response
Use WebSphere MQ Explorer to investigate why the service could not begin. If recovery for this service is active, MQ attempts to recover.

AMQ7881 (Windows)
Unable to stop <*insert_4*>/<*insert_3*> WebSphere MQ service, return code <*insert_1*>.

Severity
10 : Warning

Explanation
The WebSphere MQ service was unable to stop <*insert_4*>/<*insert_3*>. The error message reported was as follows: <*insert_5*>

Response
Use WebSphere MQ Explorer to investigate why the service could not be stopped.

AMQ7882 (Windows)
Attempting to recover <*insert_4*>/<*insert_3*> WebSphere MQ service.

Severity
0 : Information

Explanation
The WebSphere MQ service has detected that <*insert_4*>/<*insert_3*> has failed, and is attempting to restart it.

Response
No Action Required.

AMQ7883 (Windows)
<*insert_4*>/<*insert_3*> WebSphere MQ service started from recovery.

Severity
0 : Information

Explanation
The WebSphere MQ service has successfully recovered <*insert_4*>/<*insert_3*>.

Response
No Action Required.

AMQ7884 (Windows)
Unable to recover <*insert_4*>/<*insert_3*> WebSphere MQ service.

Severity
10 : Warning

Explanation
The WebSphere MQ service has attempted to recover <*insert_4*>/<*insert_3*>, but all attempts have failed. There are no more attempts to recover this service.

Response
Use WebSphere MQ Explorer to investigate why the service failed and could not be restarted.

AMQ7885 (Windows)
Unable to delete queue manager <*insert_4*>, error <*insert_1*>.

Severity
10 : Warning

Explanation
An attempt to delete queue manager <*insert_4*> failed. WebSphere MQ returned error code <*insert_1*>: <*insert_5*>

Response
Ensure that the queue manager name has been specified correctly, and try again.

AMQ7886 (Windows)
Unable to create queue manager <*insert_4*>.

Severity
10 : Warning

Explanation

Queue manager <*insert_4*> could not be created. WebSphere MQ returned error <*insert_1*>: <*insert_5*>

Response

Check the error and application event logs to investigate the reason for the returned error and suggested responses to take to rectify the fault. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7890 (Windows)

Unable to open mapped file containing WebSphere MQ performance data.

Severity

20 : Error

Explanation

The WebSphere MQ extensible counter dll was unable to open a mapped file used to collect queue performance data. Your system might be running short on virtual storage.

Response

No action required. Performance statistics for MQ queues will not be displayed.

AMQ7891 (Windows)

Unable to create a mutex to access WebSphere MQ performance data.

Severity

20 : Error

Explanation

The WebSphere MQ extensible counter dll was unable to create a mutex required to synchronize collection of queue performance data

Response

No action required. Performance statistics for MQ queues will not be displayed.

AMQ7892 (Windows)

Unable to map to shared memory file containing WebSphere MQ performance data.

Severity

20 : Error

Explanation

The WebSphere MQ extensible counter dll was unable to map the shared memory file required for collection of queue performance data.

Response

No action required. Performance statistics for MQ queues will not be displayed.

AMQ7893 (Windows)

Unable to open "Performance" key for WebSphere MQ services. Status code: <*insert_1*>.

Severity

20 : Error

Explanation

The WebSphere MQ extensible counter dll was unable to obtain performance counter values from the "Performance" key for WebSphere MQ services. Status code is the return value from the Windows registry call RegOpenKeyEx.

Response
No action required. Performance statistics for MQ queues will not be displayed.

AMQ7894 (Windows)
Unable to read the "Performance\First Counter" value for WebSphere MQ services. Status code: <*insert_1*>.

Severity
20 : Error

Explanation
The WebSphere MQ extensible counter dll was unable to obtain performance counter values from the "Performance\First Counter" key for WebSphere MQ services. Status code is the return value from the Windows registry call RegOpenKeyEx.

Response
No action required. Performance statistics for MQ queues will not be displayed.

AMQ7895 (Windows)
Unable to read the "Performance\First Help" value for WebSphere MQ services. Status code: <*insert_1*>.

Severity
20 : Error

Explanation
The WebSphere MQ extensible counter dll was unable to obtain performance counter values from the "Performance\First Help" key for WebSphere MQ services. Status code is the return value from the Windows registry call RegOpenKeyEx.

Response
No action required. Performance statistics for MQ queues will not be displayed.

AMQ7901
The data-conversion exit <*insert_3*> has not loaded.

Severity
30 : Severe error

Explanation
The data-conversion exit program, <*insert_3*>, failed to load. The internal function gave exception <*insert_4*>.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7903
The data-conversion exit <*insert_3*> cannot be found.

Severity
30 : Severe error

Explanation
Message data conversion has been requested for a WebSphere MQ message with a user-defined format, but the necessary data-conversion exit program, <*insert_3*>, cannot be found. The internal function gave exception <*insert_4*>.

Response
Check that the necessary data-conversion exit <*insert_3*> exists.

AMQ7904

The data-conversion exit <*insert_3*> cannot be found, or loaded.

Severity

30 : Severe error

Explanation

Message data conversion was requested for a WebSphere MQ message with a user-defined format, but the necessary data conversion exit program, <*insert_3*>, was not found, or loaded. The <*insert_4*> function call gave a return code of <*insert_1*>.

Response

Check that the necessary data conversion exit routine exists in one of the standard directories for dynamically loaded modules. If necessary, inspect the generated output to examine the message descriptor (MQMD structure) of the MQ message for the conversion which was requested. This might help you to determine where the message originated.

AMQ7905

Unexpected exception <*insert_4*> in data-conversion exit.

Severity

30 : Severe error

Explanation

The data-conversion exit program, <*insert_3*>, ended with an unexpected exception <*insert_4*>. The message has not been converted.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7907

Unexpected exception in data-conversion exit.

Severity

30 : Severe error

Explanation

The data-conversion exit routine, <*insert_3*>, ended with an unexpected exception. The message has not been converted.

Response

Correct the error in the data-conversion exit routine.

AMQ7908 (Windows)

Display active directory CRL server details.

Severity

0 : Information

Explanation

Display active directory CRL server details.

Response

None.

AMQ7909 (Windows)

There are no active directory CRL server details to display.

Severity
0 : Information

Explanation
No active directory CRL server definitions could be found.

Response
None.

AMQ7910 (Windows)
Usage: setmqscp [-a [-m QmgrName | *] | -r [-m QmgrName | *] | -d]

Severity
0 : Information

AMQ7911 (Windows)
The default Active Directory could not be located on your domain.

Severity
20 : Error

Explanation
No domain controllers with Active Directories could be found on the domain that your computer is a member of.

Response
Active Directory support for MQ MQI client connections cannot be used without a default Active Directory available on your domain.

AMQ7912 (Windows)
The Active Directory support library failed to initialize.

Severity
20 : Error

Explanation
WebSphere MQ support libraries for Active Directory client connections could not be initialized.

Response
Check that the Active Directory client pre-requisite software has been installed on your machine before attempting to use this feature.

AMQ7913 (Windows)
The WebSphere MQ Active Directory container could not be created.

Severity
20 : Error

Explanation
WebSphere MQ has failed to create an IBM-MQClientConnections container as a child of your domain's system container in the Active Directory.

Response
Ensure that you have permission to create sub-containers of the system container, and modify the otherWellKnownObjects property of the system container.

AMQ7914 (Windows)
Migration of the client connection table for Queue Manager <*insert_3*> failed with reason code <*insert_1*><*insert_4*>.

Severity
10 : Warning

Explanation
The client connection table for this Queue Manager could not be migrated at this time.

Response
Ensure that the client connection table exists and is not corrupted, and that you have authority to create new objects in the Active Directory on your domain.

AMQ7915 (Windows)
Created service connection point for connection <*insert_3*>.

Severity
0 : Information

Explanation
The service connection point was successfully created for this client connection.

Response
None.

AMQ7916 (Windows)
The Active Directory channel definition table could not be opened.

Severity
20 : Error

Explanation
The IBM-MQClientConnections Active Directory container could not be located in the Global Catalog.

Response
Ensure that setmqscp has been used to create the container object and that you have permission to read the container and its child objects.

AMQ7917 (Windows)
Display active directory channel details.

Severity
0 : Information

Explanation
Display active directory channel details.

Response
None.

AMQ7918 (Windows)
The WebSphere MQ Active Directory container could not be deleted.

Severity
20 : Error

Explanation
There was a problem when attempting to delete the MQ Active Directory container. The container must be empty before it can be deleted from the directory.

Response
None.

AMQ7919 (Windows)
There are no active directory client channel details to display.

Severity
0 : Information

Explanation
No active directory client channel definitions could be found.

Response
None.

AMQ7920 (Windows)

Usage: setmqcrl [-m QmgrName] [-a] [-d] [-r]

Severity

0 : Information

AMQ7921

An incorrect eye-catcher field in an MQDXP structure has been detected.

Severity

30 : Severe error

Explanation

The MQDXP structure passed to the Internal Formats Conversion routine contains an incorrect eye-catcher field.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ7922

A PCF message is incomplete.

Severity

30 : Severe error

Explanation

Message data conversion cannot convert a message in Programmable Command Format (PCF) because the message is only <*insert_1*> bytes long and does not contain a PCF header. The message has either been truncated, or it contains data that is not valid.

Response

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7923

A message had an unrecognized integer encoding - <*insert_1*>.

Severity

30 : Severe error

Explanation

Message data conversion cannot convert a message because the integer encoding value of the message, <*insert_1*>, was not recognized.

Response

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7924

Bad length in the PCF header (length = <*insert_1*>).

Severity

30 : Severe error

Explanation

Message data conversion cannot convert a message in Programmable Command Format (PCF) because the PCF header structure contains an incorrect length field. Either the message has been truncated, or it contains data that is not valid.

Response

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7925

Message version <*insert_1*> is not supported.

Severity

30 : Severe error

Explanation

Message data conversion cannot convert a message because the Version field of the message contains an incorrect value.

Response

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7926

A PCF message has an incorrect parameter count value <*insert_1*>.

Severity

30 : Severe error

Explanation

Message data conversion cannot convert a message in Programmable Command Format (PCF) because the parameter count field of the PCF header is incorrect.

Response

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7927

Bad type in PCF structure number <*insert_1*> (type = <*insert_2*>).

Severity

30 : Severe error

Explanation

A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contained an incorrect type field.

Response

Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7928

Bad length in PCF structure number <*insert_1*> (length = <*insert_2*>).

Severity
30 : Severe error

Explanation
A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contained an incorrect length field.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7929
A PCF structure is incomplete.

Severity
30 : Severe error

Explanation
Message data conversion cannot convert a message in Programmable Command Format (PCF) because structure number <*insert_1*>, of Type value <*insert_2*>, within the message is incomplete. The message has either been truncated, or it contains data that is not valid.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7930
Bad CCSID in PCF structure number <*insert_1*> (CCSID = <*insert_2*>).

Severity
30 : Severe error

Explanation
A Programmable Command Format (PCF) structure passed to the Internal Formats Converter contains an incorrect CCSID.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7931
Bad length in PCF structure number <*insert_1*> (length = <*insert_2*>).

Severity
30 : Severe error

Explanation
Message data conversion cannot convert a message in Programmable Command Format (PCF) because one of the structures of the message contains an incorrect length field.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7932
Bad count in PCF structure number <*insert_1*> (count = <*insert_2*>).

Severity
30 : Severe error

Explanation
Message data conversion cannot convert a message in Programmable Command Format (PCF) because a StringList structure of the message contains an incorrect count field.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message, and the incorrect structure to determine the source of the message, and to see how data that is not valid became included in the message.

AMQ7933
Bad string length in PCF structure.

Severity
30 : Severe error

Explanation
Message data conversion cannot convert a message in Programmable Command Format (PCF) because structure number <*insert_1*> of the message contains an incorrect string length value <*insert_2*>.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message, and the incorrect structure to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7934
Wrong combination of MQCCSI_DEFAULT with MQCCSI_EMBEDDED or MQEPH_CCSID_EMBEDDED.

Severity
30 : Severe error

Explanation
Message data conversion could not convert a message in Programmable Command Format (PCF) because structure <*insert_1*> of the message contained a CodedCharSetId field of MQCCSI_DEFAULT while the message itself had a CodedCharSetId of MQCCSI_EMBEDDED, or the Flags field of the MQEPH structure containing the PCF specified flag MQEPH_CCSID_EMBEDDED. These are incorrect combinations.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor, the headers of the message and the incorrect structure to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7935
Bad CCSID in message header (CCSID = <*insert_1*>).

Severity
30 : Severe error

Explanation
Message data conversion could not convert a message because the Message Descriptor of the message contained an incorrect CodedCharSetId field.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Do not discard these files until the problem has been resolved. Use the file containing the Message Descriptor of the message to determine the source of the message and to see how data that is not valid became included in the message.

AMQ7936
The file <*insert_3*> already exists.

Severity
30 : Severe error

Explanation
The output file already exists, but REPLACE has not been specified.

Response
Specify REPLACE to over-write the existing file, or select a different output file name.

AMQ7937
Structure length <*insert_1*> in MQFMT_IMS_VAR_STRING format message is not valid.

Severity
30 : Severe error

Explanation
This error is detected when attempting data conversion. The valid range for the length is 4 (with no string data) to 32767. The message is returned unconverted with a reason code of MQRC_CONVERTED_STRING_TOO_BIG.

Response
Check the content of the message before data conversion and correct the message format. When converting data using two or more bytes per character, remember that the number of bytes in each character can change during data conversion. This causes the message lengths to change.

AMQ7943
Usage: crtmqcvx SourceFile TargetFile

Severity
0 : Information

Explanation
None.

Response
None.

AMQ7953
One structure has been parsed.

Severity
0 : Information

Explanation
The crtmqcvx command has parsed one structure.

Response
None.

AMQ7954
<*insert_1*> structures have been parsed.

Severity
0 : Information

Explanation
The crtmqcvx command has parsed <*insert_1*> structures.

Response
None.

AMQ7955
Unexpected field: <*insert_1*>.

Severity
0 : Information

Explanation
The field within the structure is of a type that is not recognized.

Response
Correct the field and retry the command.

AMQ7956
Bad array dimension.

Severity
0 : Information

Explanation
An array field of the structure has an incorrect dimension value.

Response
Correct the field and retry the command.

AMQ7957
Warning at line <*insert_1*>.

Severity
20 : Error

Explanation
The structure contains another field after a variable length field. A variable length field must be the last field of the structure.

Response
Correct the structure and retry the command.

AMQ7958
Error at line <*insert_1*> in field <*insert_3*>.

Severity
30 : Severe error

Explanation
Field name <*insert_3*> is a field of type 'float'. Fields of type float are not supported by this command.

Response
Either correct the structure to eliminate fields of type float, or write your own routine to support conversion of these fields.

AMQ7959
Error at line <*insert_1*> in field <*insert_3*>.

Severity
30 : Severe error

Explanation
Field name <*insert_3*> is a field of type 'double'. Fields of type double are not supported by this command.

Response
Either correct the structure to eliminate fields of type double, or write your own routine to support conversion of these fields.

AMQ7960
Error at line <*insert_1*> in field <*insert_3*>.

Severity
30 : Severe error

Explanation
Field name <*insert_3*> is a 'pointer' field. Fields of type pointer are not supported by this command.

Response
Either correct the structure to eliminate fields of type pointer, or write your own routine to support conversion of these fields.

AMQ7961
Error at line <*insert_1*> in field <*insert_3*>.

Severity
30 : Severe error

Explanation
Field name <*insert_3*> is a 'bit' field. Bit fields are not supported by this command.

Response
Either correct the structure to eliminate bit fields, or write your own routine to support conversion of these fields.

AMQ7962
No input file specified.

Severity
30 : Severe error

Explanation
This command requires that an input file is specified.

Response
Specify the name of the input file and retry the command.

AMQ7963
No output file specified.

Severity
30 : Severe error

Explanation
This command requires that an output file name is specified.

Response
Specify the name of the output file and retry the command.

AMQ7964
Unexpected option <*insert_3*>.

Severity
30 : Severe error

Explanation
The option specified is not valid for this command.

Response
Retry the command with a valid option.

AMQ7965
Incorrect number of arguments.

Severity
30 : Severe error

Explanation
The command was passed an incorrect number of arguments.

Response
Retry the command, passing it the correct number of arguments.

AMQ7968
Cannot open file <*insert_3*>.

Severity
30 : Severe error

Explanation
You cannot open the file <*insert_3*>.

Response
Check that you have the correct authorization to the file and retry the command.

AMQ7969
Syntax error.

Severity
30 : Severe error

Explanation
This line of the input file contains a language syntax error.

Response
Correct the syntax error and retry the command.

AMQ7970
Syntax error on line <*insert_1*>.

Severity
30 : Severe error

Explanation
This message identifies where, in the input file, a previously reported error was detected.

Response
Correct the error and retry the command.

AMQ7985 (Windows)
The WebSphere MQ Active Directory container already exists.

Severity
0 : Information

Explanation
The IBM-MQClientConnections Active Directory container already exists and does not need to be re-created.

Response
None.

AMQ7986 (Windows)
The WebSphere MQ Active Directory container was successfully created.

Severity
0 : Information

Explanation
The IBM-MQClientConnections Active Directory container was successfully created.

Response
None.

AMQ7987 (Windows)
Removed service connection point for connection <*insert_3*>.

Severity
0 : Information

Explanation
The service connection point was successfully removed for this client connection.

Response
None.

AMQ7988 (Windows)
Failure removing service connection point for connection <*insert_3*>.

Severity
10 : Warning

Explanation
The service connection point could not be removed for this client connection.

Response
None.

AMQ7989 (Windows)
The WebSphere MQ Active Directory container was removed successfully.

Severity
0 : Information

Explanation
The IBM-MQClientConnections Active Directory container was removed successfully.

Response
None.

AMQ7990 (Windows)
The WebSphere MQ Active Directory container does not exist.

Severity
0 : Information

Explanation
The IBM-MQClientConnections Active Directory container does not exist.

Response
None.

AMQ7A01 (IBM i)
Convert MQ Data Type

AMQ7A02 (IBM i)
Display MQ Version

AMQ7A03 (IBM i)
Create MQ Listener

AMQ7A04 (IBM i)
Listener name

AMQ7A05 (IBM i)
Listener control

AMQ7A06 (IBM i)
Listener backlog

AMQ7A07 (IBM i)
Change MQ Listener

AMQ7A08 (IBM i)
Copy MQ Listener

AMQ7A09 (IBM i)
From Listener

AMQ7A0A (IBM i)
To Listener

AMQ7A0B (IBM i)
Display MQ Listener

AMQ7A0C (IBM i)
Delete MQ Listener

AMQ7A0D (IBM i)
LSRNAME not allowed with PORT

Severity
40 : Stop Error

Explanation
A listener object cannot be specified with a port.

Response
Specify either a listener object or a port number.

AMQ7A0E (IBM i)
LSRNAME not allowed with IPADDR

Severity
40 : Stop Error

Explanation
A listener object cannot be specified with an IP address.

Response
Specify either a listener object or an IP address.

AMQ7A0F (IBM i)
Work with MQ Listener object

AMQ7A10 (IBM i)
Create MQ Service

AMQ7A11 (IBM i)
Change MQ Service

AMQ7A12 (IBM i)
Copy MQ Service

AMQ7A13 (IBM i)
Service name

AMQ7A14 (IBM i)
Start program

AMQ7A15 (IBM i)
Start program arguments

AMQ7A16 (IBM i)
End program

AMQ7A17 (IBM i)
End program arguments

AMQ7A18 (IBM i)
Standard output

AMQ7A19 (IBM i)
Standard error

AMQ7A1A (IBM i)
Service type

AMQ7A1B (IBM i)
Service control

AMQ7A1C (IBM i)
From Service

AMQ7A1D (IBM i)
To Service

AMQ7A1E (IBM i)
Display MQ Service

AMQ7A1F (IBM i)
Permit Standby Queue Manager

AMQ7A20 (IBM i)
Delete MQ Service

AMQ7A21 (IBM i)
Work with MQ Service object

AMQ7A23 (IBM i)
Start MQ Service

AMQ7A24 (IBM i)
End MQ Service

AMQ7A25 (IBM i)
Channel initiator control

AMQ7A26 (IBM i)
Command server control

AMQ7A27 (IBM i)
Display Queue Manager Status

AMQ7A28 (IBM i)
Display Listener Status

AMQ7A29 (IBM i)
Display Service Status

AMQ7A2A (IBM i)
LSRNAME not allowed with OPTION

Severity
40 : Stop Error

Explanation
A listener object cannot be specified with an end option.

Response
Specify either a listener object or an end option.

AMQ7A2B (IBM i)
Service startup

AMQ7A2C (IBM i)
Work with Connection Handles

AMQ7A2D (IBM i)
Connection Identifier

AMQ7A2E (IBM i)
End Queue Manager Connection

AMQ7A2F (IBM i)
Work with MQ Connections

AMQ7A30 (IBM i)
Header Compression

AMQ7A31 (IBM i)
Message Compression

AMQ7A32 (IBM i)
Message compression *ANY not valid for channel type.

Severity
30 : Severe error

Explanation
The message compression value *ANY is only valid for *RCVR, *RQSTR and *SVRCN channel types.

Response
Specify a valid message compression list.

AMQ7A33 (IBM i)
Channel Monitoring

AMQ7A34 (IBM i)
Channel Statistics

AMQ7A35 (IBM i)
Cluster Workload Rank

AMQ7A36 (IBM i)
Cluster Workload Priority

AMQ7A37 (IBM i)
Cluster Channel Weight

AMQ7A38 (IBM i)
Cluster workload channels

AMQ7A39 (IBM i)
Cluster workload queue use

AMQ7A3A (IBM i)
Queue Monitoring

AMQ7A3B (IBM i)
Queue Manager Statistics

AMQ7A3C (IBM i)
Cluster Sender Monitoring

AMQ7A3D (IBM i)
Queue Statistics

AMQ7A3E (IBM i)
Cluster Sender Statistics

AMQ7A3F (IBM i)
Statistics Interval

AMQ7A40 (IBM i)
Display MQ Route Information

AMQ7A41 (IBM i)
Correlation Identifier

AMQ7A42 (IBM i)
Message Persistence

AMQ7A43 (IBM i)
Message Priority

AMQ7A44 (IBM i)
Report Option

AMQ7A45 (IBM i)
Reply Queue

AMQ7A46 (IBM i)
Reply Queue Manager

AMQ7A47 (IBM i)
Message Expiry

AMQ7A48 (IBM i)
Pass Expiry

AMQ7A49 (IBM i)
Route Accumulation

AMQ7A4A (IBM i)
Reply Message

AMQ7A4B (IBM i)
Deliver Message

AMQ7A4C (IBM i)
Forward Message

AMQ7A4D (IBM i)
Maximum Activities

AMQ7A4E (IBM i)
Route Detail

AMQ7A4F (IBM i)
Browse Only

AMQ7A50 (IBM i)
Display Message

AMQ7A51 (IBM i)
Target Queue Manager

AMQ7A52 (IBM i)
Display Information

AMQ7A53 (IBM i)
Wait Time

AMQ7A54 (IBM i)
RTEINF(*YES) required for RPLYMSG(*YES).

Severity
30 : Severe error

Explanation
RPLYMSG(*YES) cannot be specified without RTEINF(*YES).

Response
If RPLYMSG(*YES) is specified then RTEINF(*YES) must also be specified.

AMQ7A55 (IBM i)
RPLYQ required for RPLYMQM.

Severity
30 : Severe error

Explanation
RPLYMQM cannot be specified without RPLYQ.

Response
If RPLYMQM is specified then RPLYQ must also be specified.

AMQ7A56 (IBM i)
CRRLID specified with invalid parameters.

Severity
30 : Severe error

Explanation
The CRRLID parameter was specified with one or more of MSGPST, MSGPRTY, OPTION, RPLYQ, RPLYMQM, EXPIRY, EXPRPT, RTEINF RPLYMSG, DLVRMSG, FWDMSG, MAXACTS, DETAIL and BIND which are invalid with CRRLID.

Response
Specify only those parameters which are valid with CRRLID.

AMQ7A57 (IBM i)
DSPMSG(*NO) specified with invalid parameters.

Severity
30 : Severe error

Explanation
DSPMSG(*NO) was specified with one or more of BROWSE, DSPINF and WAIT which are invalid with DSPMSG(*NO).

Response
Specify only those parameters which are valid with DSPMSG(*NO).

AMQ7A58 (IBM i)
RPLYQ required for DSPMSG(*NO) and RPLYMSG(*YES).

Severity
30 : Severe error

Explanation
DSPMSG(*NO) and RPLYMSG(*YES) cannot be specified without RPLYQ.

Response
If DSPMSG(*NO) and RPLYMSG(*YES) are specified than RPLYQ must also be specified.

AMQ7A59 (IBM i)
RPLYQ required for DSPMSG(*NO) and OPTION not *NONE.

Severity
30 : Severe error

Explanation
DSPMSG(*NO) and OPTION not *NONE cannot be specified without RPLYQ.

Response
If DSPMSG(*NO) and OPTION not *NONE are specified than RPLYQ must also be specified.

AMQ7A5A (IBM i)
Run WebSphere MQ Commands

AMQ7A5B (IBM i)
Non Persistent Message Class

AMQ7A5C (IBM i)
NPMCLASS not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The NPMCLASS parameter cannot be specified for a queue of type *ALS or *RMT.

Response
Remove the NPMCLASS parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7A5D (IBM i)
MONQ not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The MONQ parameter cannot be specified for a queue of type *ALS or *RMT.

Response
Remove the MONQ parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7A5E (IBM i)
STATQ not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The STATQ parameter cannot be specified for a queue of type *ALS or *RMT.

Response
Remove the STATQ parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7A5F (IBM i)
ACCTQ not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The ACCTQ parameter cannot be specified for a queue of type *ALS or *RMT.

Response
Remove the ACCTQ parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7A60 (IBM i)
All queue managers have been quiesced.

Severity
0 : Information

Explanation
All queue managers have been successfully quiesced.

Response
None.

AMQ7A61 (IBM i)
MQMNAME not valid for TRCEARLY(*YES).

Severity
40 : Stop Error

Explanation
The MQMNAME parameter can only be specified for TRCEARLY(*NO). TRCEARLY(*YES) applies to all queue managers.

Response
If TRCEARLY(*YES) is required remove MQMNAME from the command.

AMQ7A62 (IBM i)
MQMNAME not valid for SET(*END).

Severity
40 : Stop Error

Explanation
The MQMNAME parameter can only be specified for SET(*ON) or SET(*OFF). SET(*END) applies to all queue managers.

Response
If SET(*END) is required remove MQMNAME from the command.

AMQ7A63 (IBM i)
Bind Option

AMQ7A64 (IBM i)
TGTMQMNAME only valid for channel type *CLTCN.

Severity
40 : Stop Error

Explanation
The TGTMQMNAME parameter can only be specified with channel type *CLTCN.

Response
Remove the TGTMQMNAME parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7A65 (IBM i)
Invalid value specified for JOB parameter.

Severity
40 : Stop Error

Explanation
A value for the JOB parameter has been specified however the format of the parameter is incorrect. The value of this parameter can be one of the followng formats:

generic-jobname

Job-name/User/Number

Job-name/User/Number/thread-identifier.

Note that the thread-identifier cannot be specified without a fully qualified jobname.

Response
Specify a value in one of the acceptable formats and then try the command again. If you are prompting this command, you must enter characters in the job name field first to clear an invalid value specified elsewhere in the parameter entry.

AMQ7A66 (IBM i)
Data Directory Prefix

AMQ7A67 (IBM i)
IPC Directory Prefix

AMQ7A68 (IBM i)
Allow Switchover

AMQ7A69 (IBM i)
ASP device

AMQ7B00 (IBM i)
MQI Accounting

AMQ7B01 (IBM i)
Input file

AMQ7B02 (IBM i)
Queue Accounting

AMQ7B03 (IBM i)
Member containing input

AMQ7B04 (IBM i)
Accounting Interval

AMQ7B05 (IBM i)
Accounting Override

AMQ7B06 (IBM i)
Trace data size

AMQ7B07 (IBM i)
Perform replay only

AMQ7B08 (IBM i)
Activate backup

AMQ7B09 (IBM i)
No connection handles to display

AMQ7B0A (IBM i)
Trace Route Recording

AMQ7B0B (IBM i)
Activity Recording

AMQ7B0C (IBM i)
No queue manager connections to display

AMQ7B0D (IBM i)
No listener objects to display

AMQ7B0E (IBM i)
No service objects to display

AMQ7B0F (IBM i)
CLWLRANK not allowed with queue type *MDL.

Severity
40 : Stop Error

Explanation
The CLWLRANK parameter cannot be specified for a queue of type *MDL.

Response
Remove the CLWLRANK parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7B10 (IBM i)
CLWLPRTY not allowed with queue type *MDL.

Severity
40 : Stop Error

Explanation
The CLWLPRTY parameter cannot be specified for a queue of type *MDL.

Response
Remove the CLWLPRTY parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7B11 (IBM i)
LSRNAME not allowed with BACKLOG

Severity
40 : Stop Error

Explanation
A listener object cannot be specified with a listener backlog.

Response
Specify either a listener object or a listener backlog.

AMQ7B12 (IBM i)
MONCHL not valid for channel type *CLTCN.

Severity
40 : Stop Error

Explanation
The MONCHL parameter cannot be specified with channel type *CLTCN.

Response
Remove the MONCHL parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B13 (IBM i)
STATCHL not valid for channel types *CLTCN and *SVRCN

Severity
40 : Stop Error

Explanation
The STATCHL parameter is valid only with channel type *SDR, *SVR, *RCVR, *RQSTR, *CLUSSDR or *CLUSRCVR.

Response
Remove the STATCHL parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B14 (IBM i)
CLWLRANK only valid for channel types *CLUSSDR and *CLUSRCVR.

Severity
40 : Stop Error

Explanation
The CLWLRANK parameter can only be specified with channel types *CLUSSDR or *CLUSRCVR.

Response
Remove the CLWLRANK parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B15 (IBM i)
CLWLPRTY only valid for channel types *CLUSSDR and *CLUSRCVR.

Severity
40 : Stop Error

Explanation
The CLWLPRTY parameter can only be specified with channel types *CLUSSDR or *CLUSRCVR.

Response
Remove the CLWLPRTY parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B16 (IBM i)
CLWLWGHT only valid for channel types *CLUSSDR and *CLUSRCVR.

Severity
40 : Stop Error

Explanation
The CLWLWGHT parameter can only be specified with channel types *CLUSSDR or *CLUSRCVR.

Response
Remove the CLWLWGHT parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B17 (IBM i)
CLWLUSEQ only allowed with queue type *LCL.

Severity
40 : Stop Error

Explanation
The CLWLUSEQ parameter can only be specified for a queue of type *LCL.

Response
Remove the CLWLUSEQ parameter from the command or, if the command is CRTMQMQ, specify a value of *LCL for QTYPE. Then try the command again.

AMQ7B18 (IBM i)
MCAUSRID not valid for channel type *CLTCN.

Severity
40 : Stop Error

Explanation
The MCAUSRID parameter cannot be specified with channel type *CLTCN.

Response
Remove the MCAUSRID parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B20 (IBM i)
Message Read Ahead

AMQ7B21 (IBM i)
MSGREADAHD not allowed with queue type *RMT.

Severity
40 : Stop Error

Explanation
The MSGREADAHD parameter cannot be specified for a queue of type *RMT.

Response
Remove the MSGREADAHD parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7B22 (IBM i)
Sharing Conversations

AMQ7B23 (IBM i)
SHARECNV is valid only when CHLTYPE is *SVRCN or *CLTCN.

Severity
40 : Stop Error

Explanation
The sharing conversations (SHARECNV) parameter cannot be specified for a channel type other than *SVRCN or *CLTCN.

Response
Remove the SHARECNV parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B24 (IBM i)
Maximum Property Data Length

AMQ7B25 (IBM i)
Default Put Response

AMQ7B26 (IBM i)
Message mark-browse interval

AMQ7B27 (IBM i)
Property Control

AMQ7B28 (IBM i)
Maximum Instances

AMQ7B29 (IBM i)
Maximum Instances Per Client

AMQ7B2A (IBM i)
Client Channel Weight

AMQ7B2B (IBM i)
Connection Affinity

AMQ7B2C (IBM i)
Target Type

AMQ7B2D (IBM i)
PROPCTL not allowed with queue type *RMT.

Severity
40 : Stop Error

Explanation
The PROPCTL parameter cannot be specified for a queue of type *RMT.

Response
Remove the PROPCTL parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7B2E (IBM i)
TARGTYPE only allowed with queue type *ALS.

Severity
40 : Stop Error

Explanation
The TARGTYPE parameter can only be specified for a queue of type *ALS.

Response
Remove the TARGTYPE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ7B2F (IBM i)
PROPCTL only allowed with channel type *SDR, *SRV, *CLUSSDR or *CLUSRCVR.

Severity
40 : Stop Error

Explanation
The PROPCTL parameter can only be specified for a channel of type *SDR, *SVR, *CLUSSDR or *CLUSRCVR.

Response
Remove the PROPCTL parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B30 (IBM i)
MAXINST only allowed with channel type *SVRCN.

Severity
40 : Stop Error

Explanation
The MAXINST parameter can only be specified for a channel of type *SVRCN.

Response
Remove the MAXINST parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B31 (IBM i)
MAXINSTC only allowed with channel type *SVRCN.

Severity
40 : Stop Error

Explanation
The MAXINSTC parameter can only be specified for a channel of type *SVRCN.

Response
Remove the MAXINSTC parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B32 (IBM i)
CLNTWGHT only allowed with channel type *CLTCN.

Severity
40 : Stop Error

Explanation
The CLNTWGHT parameter can only be specified for a channel of type *CLTCN.

Response
Remove the CLNTWGHT parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B33 (IBM i)
AFFINITY only allowed with channel type *CLTCN.

Severity
40 : Stop Error

Explanation
The AFFINITY parameter can only be specified for a channel of type *CLTCN.

Response
Remove the AFFINITY parameter from the command or, if the command is CRTMQMCHL, specify a different value for CHLTYPE. Then try the command again.

AMQ7B34 (IBM i)
Create MQ Topic

AMQ7B35 (IBM i)
Change MQ Topic

AMQ7B36 (IBM i)
Copy MQ Topic

AMQ7B37 (IBM i)
Display MQ Topic

AMQ7B38 (IBM i)
Topic name

AMQ7B39 (IBM i)
Topic string

AMQ7B3A (IBM i)
Durable subscriptions

AMQ7B3B (IBM i)
Durable model queue

AMQ7B3C (IBM i)
Non-durable model queue

AMQ7B3D (IBM i)
Publish

AMQ7B3E (IBM i)
Subscribe

AMQ7B3F (IBM i)
Wildcard behaviour

AMQ7B40 (IBM i)
Persistent message delivery

AMQ7B41 (IBM i)
Non-persistent message delivery

AMQ7B42 (IBM i)
From topic

AMQ7B43 (IBM i)
To topic

AMQ7B44 (IBM i)
PubSub max msg retry count

AMQ7B45 (IBM i)
PubSub NPM msg

AMQ7B46 (IBM i)
PubSub NPM msg response

AMQ7B47 (IBM i)
PubSub syncpoint

AMQ7B48 (IBM i)
Change MQ Subscription

AMQ7B49 (IBM i)
Copy MQ Subscription

AMQ7B4A (IBM i)
From subscription

AMQ7B4B (IBM i)
To subscription

AMQ7B4C (IBM i)
Destination Queue Manager

AMQ7B4D (IBM i)
Destination Correlation Id

AMQ7B4E (IBM i)
Subscription User Id

AMQ7B4F (IBM i)
Publish Application Id

AMQ7B50 (IBM i)
Subscription User Data

AMQ7B51 (IBM i)
Selector String

AMQ7B52 (IBM i)
PubSub Property

AMQ7B53 (IBM i)
Destination Class

AMQ7B54 (IBM i)
Subscription Scope

AMQ7B55 (IBM i)
Variable User

AMQ7B57 (IBM i)
Request Publications

AMQ7B58 (IBM i)
Publish Priority

AMQ7B59 (IBM i)
Wildcard Schema

AMQ7B5A (IBM i)
Expiry Time

AMQ7B5B (IBM i)
Create MQ Subscription

AMQ7B5C (IBM i)
Subscription name

AMQ7B5D (IBM i)
Topic object

AMQ7B5E (IBM i)
Destination

AMQ7B5F (IBM i)
Work with MQ Subscriptions

AMQ7B60 (IBM i)
No subscriptions to display

AMQ7B61 (IBM i)
Display MQ Subscription

AMQ7B62 (IBM i)
Delete MQ Subscription

AMQ7B63 (IBM i)
Publish Accounting Token

AMQ7B67 (IBM i)
Subscription identifier

AMQ7B68 (IBM i)
From subscription identifier

AMQ7B69 (IBM i)
Pubsub Engine Control

AMQ7B6A (IBM i)
No message properties to display.

Severity
0 : Information

Explanation
The message contains no message properties.

Response
None.

AMQ7B6B (IBM i)
Trace directory

AMQ7B6C (IBM i)
Trace start control

AMQ7B6D (IBM i)
User

AMQ7B6E (IBM i)
Trace end control

AMQ7B6F (IBM i)
Clear MQ Topic String

AMQ7B71 (IBM i)
Topic Tree Life Time

AMQ7B72 (IBM i)
Job information

AMQ7B73 (IBM i)
Thread identifier

AMQ7B74 (IBM i)
Clear type

AMQ7B75 (IBM i)
Clear scope

AMQ7B76 (IBM i)
Invalid combination of security exit parameters.

Severity
40 : Stop Error

Explanation
An invalid combination of security exit parameters has been provided on the command. The SCYEXIT parameter cannot be specified for a channel of type *CLTCN. The CSCYEXIT parameter can only be specified for a channel of type *CLTCN. You cannot specify both SCYEXIT and CSCYEXIT parameters together on the same command.

Response
Remove the invalid combination of security exit parameters from the command and then try the command again.

AMQ7B77 (IBM i)
Invalid combination of send exit parameters.

Severity
40 : Stop Error

Explanation
An invalid combination of send exit parameters has been provided on the command. The SNDEXIT parameter cannot be specified for a channel of type *CLTCN. The CSNDEXIT parameter can only be specified for a channel of type *CLTCN. You cannot specify both SNDEXIT and CSNDEXIT parameters together on the same command.

Response
Remove the invalid combination of send exit parameters from the command and then try the command again.

AMQ7B78 (IBM i)
Invalid combination of receive exit parameters.

Severity
40 : Stop Error

Explanation
An invalid combination of receive exit parameters has been provided on the command. The RCVEXIT parameter cannot be specified for a channel of type *CLTCN. The CRCVEXIT parameter can only be specified for a channel of type *CLTCN. You cannot specify both RCVEXIT and CRCVEXIT parameters together on the same command.

Response
Remove the invalid combination of receive exit parameters from the command and then try the command again.

AMQ7B79 (IBM i)
Command is not applicable to WebSphere MQ Publish/Subscribe broker.

Severity
0 : Information

Explanation
This command performs a null operation.

Response
Refer to Publish/Subscribe User's Guide publication for alternative ways to perform this function.

AMQ8000-8999: Administration

AMQ8001
IBM WebSphere MQ queue manager created.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_5*> created.

Response
None.

AMQ8002
IBM WebSphere MQ queue manager <*insert_5*> deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_5*> deleted.

Response
None.

AMQ8003
IBM WebSphere MQ queue manager <*insert_5*> started.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_5*> started.

Response
None.

AMQ8004
IBM WebSphere MQ queue manager <*insert_5*> ended.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_5*> ended.

Response
None.

AMQ8005
IBM WebSphere MQ queue manager changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_3*> changed.

Response
None.

AMQ8006
IBM WebSphere MQ queue created.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue <*insert_3*> created.

Response
None.

AMQ8007
IBM WebSphere MQ queue deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue <*insert_3*> deleted.

Response
None.

AMQ8008
IBM WebSphere MQ queue changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue <*insert_3*> changed.

Response
None.

AMQ8010
IBM WebSphere MQ process created.

Severity
0 : Information

Explanation
IBM WebSphere MQ process <*insert_3*> created.

Response
None.

AMQ8011
IBM WebSphere MQ process deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ process <*insert_3*> deleted.

Response
None.

AMQ8012
IBM WebSphere MQ process changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ process <*insert_3*> changed.

Response
None.

AMQ8014
IBM WebSphere MQ channel created.

Severity
0 : Information

Explanation
IBM WebSphere MQ channel <*insert_3*> created.

Response
None.

AMQ8015
IBM WebSphere MQ channel deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ channel <*insert_3*> deleted.

Response
None.

AMQ8016
IBM WebSphere MQ channel changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ channel <*insert_3*> changed.

Response
None.

AMQ8018
Start IBM WebSphere MQ channel accepted.

Severity
0 : Information

Explanation
The channel <*insert_3*> is being started. The start channel function has been initiated. This involves a series of operations across the network before the channel is actually started. The channel status displays "BINDING" for a short period while communication protocols are negotiated with the channel with which communication is being initiated.

Response
None.

AMQ8019
Stop IBM WebSphere MQ channel accepted.

Severity
0 : Information

Explanation
The channel <*insert_3*> has been requested to stop.

Response
None.

AMQ8020
Ping IBM WebSphere MQ channel complete.

Severity
0 : Information

Explanation
Ping channel <*insert_3*> complete.

Response
None.

AMQ8021
Request to start IBM WebSphere MQ Listener accepted.

Severity
0 : Information

Explanation
The Request to start the Listener has been accepted and is being processed.

Response
Should the request to start the listener be unsuccessful then information related to the error will be available in the queue manager error log. Once started the status of the listener may be monitored using the MQSC command 'DISPLAY LSSTATUS'. On IBM i the status of the listener may also be monitored using the 'WRKMQMLSR OPTION(*STATUS)' command.

AMQ8022
IBM WebSphere MQ queue cleared.

Severity
0 : Information

Explanation
All messages on queue <*insert_3*> have been deleted.

Response
None.

AMQ8023
IBM WebSphere MQ channel reset.

Severity
0 : Information

Explanation
Channel <*insert_3*> has been reset, the new sequence number of the channel is <*insert_1*>.

Response
None.

AMQ8024
IBM WebSphere MQ channel initiator started.

Severity
0 : Information

Explanation
The channel initiator for queue <*insert_3*> has been started.

Response
None.

AMQ8025
IBM WebSphere MQ channel resolved.

Severity
0 : Information

Explanation
In doubt messages for IBM WebSphere MQ channel <*insert_3*> have been resolved.

Response
None.

AMQ8026
End IBM WebSphere MQ queue manager accepted.

Severity
0 : Information

Explanation
A controlled stop request has been initiated for queue manager <*insert_5*>.

Response
None.

AMQ8027
IBM WebSphere MQ command server started.

Severity
0 : Information

Explanation
The command server has been started.

Response
None.

AMQ8028
IBM WebSphere MQ command server ended.

Severity
0 : Information

Explanation
The command server has been stopped.

Response
None.

AMQ8029
IBM WebSphere MQ authority granted.

Severity
0 : Information

Explanation
Authority for object <*insert_5*> granted.

Response
None.

AMQ8030
IBM WebSphere MQ authority revoked.

Severity
0 : Information

Explanation
Authority for object <*insert_3*> revoked.

Response
None.

AMQ8031 (IBM i)
Message Queue Manager connected.

Severity
0 : Information

Explanation
The message queue manager has been connected.

Response
None.

AMQ8032 (IBM i)
Message Queue Manager disconnected.

Severity
0 : Information

Explanation
The message queue manager has been disconnected.

Response
None.

AMQ8033
IBM WebSphere MQ object recreated.

Severity
0 : Information

Explanation
MQ object <*insert_5*> has been re-created from image.

Response
None.

AMQ8034
IBM WebSphere MQ object image recorded.

Severity
0 : Information

Explanation
Image of MQ object <*insert_3*> has been recorded.

Response
None.

AMQ8035
IBM WebSphere MQ Command Server Status . . : Running

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8036
IBM WebSphere MQ command server status . . : Stopping

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8037
IBM WebSphere MQ command server status . . : Starting

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8038
IBM WebSphere MQ command server status . . : Running with queue disabled

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8039
IBM WebSphere MQ command server status . . : Stopped

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8040
IBM WebSphere MQ command server ending.

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8041
The queue manager cannot be restarted or deleted because processes, that were previously connected, are still running.

Severity
40 : Stop Error

Explanation
Processes, that were connected to the queue manager the last time it was running, are still active. The queue manager cannot be restarted.

Response
Stop the processes and try to start the queue manager.

AMQ8041 (IBM i)
The queue manager cannot be restarted or deleted.

Severity
40 : Stop Error

Explanation
Jobs that were connected to the queue manager the last time it was running, are still active. The queue manager cannot be restarted or deleted.

Response
Use option 22 from WRKMQM to identify which jobs are connected to the queue manager. End the connected jobs and then retry the command.

AMQ8042
Process <*insert_1*> is still running.

Severity
0 : Information

AMQ8043
Non runtime application attempted to connect to runtime only queue manager.

Severity
0 : Information

Explanation
A non runtime application attempted to connect to a queue manager on a node where support for non runtime applications has not been installed. The connect attempt will be rejected with a reason of MQRC_ENVIRONMENT_ERROR.

Response
If the node is intended to support only runtime applications, investigate why a non runtime application has attempted to connect to the queue manager. If the node is intended to support non runtime only applications, investigate if the base option has been installed. The base option must be installed if non runtime applications are to run on this node.

AMQ8044 (Windows)
An error occurred while removing the queue manager from the Active Directory.

Severity
0 : Information

Explanation
The attempt to remove the queue manager from the Windows Active Directory failed. This may be because the appropriate entry could not be opened or modified, or the Service Control Point has already been removed.

Response
Check that your account has the authority to delete objects from the Active Directory, and that the entry has not already been deleted.

AMQ8045
WebSphere MQ channel in use.

Severity
20 : Error

Explanation
A process is either trying to delete a running telemetry channel, or to define a new telemetry channel using a port that is already in use. If the process is trying to define a new telemetry channel, the channel is defined but not started.

Response
Stop the process that is using the port, then either delete the previously-running channel, or start the newly-defined channel.

AMQ8046
Migrating objects for <*insert_3*>.

Severity
0 : Information

Response
None.

AMQ8047
Channel migration statistics : <*insert_1*> migrated. <*insert_2*> failed.

Severity
0 : Information

Explanation
Information on the number of channel objects migrated from previous versions of IBM WebSphere MQ channel definitions as well as any failures that occurred.

Response
None.

AMQ8048
Default objects statistics : <*insert_1*> created. <*insert_2*> replaced. <*insert_3*> failed.

Severity
0 : Information

Explanation
Information on the number of objects created or replaced successfully as well as any failures that occurred while creating the default objects.

Response
None.

AMQ8049
Object <*insert_4*>. Unable to create or replace.

Severity
20 : Error

Explanation
While creating or replacing the default object <*insert_4*> for IBM WebSphere MQ queue manager <*insert_5*> an error occurred. The error was due to improper authorization. The reason code is <*insert_1*>.

Response
Check this log for more details of what the problem may be. Make sure there are sufficient resources such as disk space and storage. For damaged or corrupted objects, replace these from backup objects. If all else fails, delete the queue manager <*insert_5*> using dltmqm and create it again using crtmqm.

AMQ8050
Creating or replacing default objects for <*insert_3*>.

Severity
0 : Information

Response
None.

AMQ8051
For details of the failures that occurred, please check AMQERR01.LOG.

Severity
0 : Information

Response
None.

AMQ8051 (Tandem)
For details of the failures that occurred, please check MQERRLG1.

Severity
0 : Information

Response
None.

AMQ8052
Completing setup.

Severity
0 : Information

Response
None.

AMQ8053
Object <*insert_4*>. Unable to create or replace.

Severity
20 : Error

Explanation
While creating or replacing the default object <*insert_4*> for IBM WebSphere MQ queue manager <*insert_5*> an error occurred. The error was due to a broken connection. The reason code is <*insert_1*>.

Response
Check this log for more details of what the problem may be. Make sure there is sufficient

resources such as disk space and storage. For damaged or corrupted objects, replace these from backup objects. If all else fails, delete the queue manager <*insert_5*> using dltmqm and create it again using crtmqm.

AMQ8054

Object <*insert_4*>. Unable to create or replace.

Severity

20 : Error

Explanation

While creating or replacing the default object <*insert_4*> for IBM WebSphere MQ queue manager <*insert_5*> an error occurred. The error was due to unavailable storage. The reason code is <*insert_1*>.

Response

Check this log for more details of what the problem may be. Make sure there is sufficient resources such as disk space and storage. For damaged or corrupted objects, replace these from backup objects. If all else fails, delete the queue manager <*insert_5*> using dltmqm and create it again using crtmqm.

AMQ8055

Object <*insert_4*>. Unable to create or replace.

Severity

20 : Error

Explanation

While creating or replacing the default object <*insert_4*> for IBM WebSphere MQ queue manager <*insert_5*> an error occurred. The error was due to a damaged object. The reason code is <*insert_1*>.

Response

Check this log for more details of what the problem may be. Make sure there is sufficient resources such as disk space and storage. For damaged or corrupted objects, replace these from backup objects. If all else fails, delete the queue manager <*insert_5*> using dltmqm and create it again using crtmqm.

AMQ8056

Object <*insert_4*>. Unable to create or replace.

Severity

20 : Error

Explanation

While creating or replacing the default object <*insert_4*> for IBM WebSphere MQ queue manager <*insert_5*> an error occurred. The error was due to a channel definition error. The error code is <*insert_1*> (X<*insert_2*>).

Response

Check this log for more details of what the problem may be. Make sure there is sufficient resources such as disk space and storage. For damaged or corrupted objects, replace these from backup objects. If all else fails, delete the queue manager <*insert_5*> using dltmqm and create it again using crtmqm.

AMQ8057

Object <*insert_4*>. Unable to create or replace.

Severity

20 : Error

Explanation

While creating or replacing the default object <*insert_4*> for IBM WebSphere MQ queue manager <*insert_5*> an error occurred. The error was due to invalid records in the channel definition file. The error code is <*insert_1*> (X<*insert_2*>).

Response

Check this log for more details of what the problem may be. Make sure there is sufficient resources such as disk space and storage. For damaged or corrupted objects, replace these from backup objects. If all else fails, delete the queue manager <*insert_5*> using dltmqm and create it again using crtmqm.

AMQ8058

Object <*insert_4*>. Unable to create or replace.

Severity

20 : Error

Explanation

While creating or replacing the default object <*insert_4*> for IBM WebSphere MQ queue manager <*insert_5*> an error occurred. The error was due to not finding the channel definition file. The error code is <*insert_1*> (X<*insert_2*>).

Response

Check this log for more details of what the problem may be. Make sure there is sufficient resources such as disk space and storage. For damaged or corrupted objects, replace these from backup objects. If all else fails, delete the queue manager <*insert_5*> using dltmqm and create it again using crtmqm.

AMQ8059

Object <*insert_4*>. Unable to create or replace.

Severity

20 : Error

Explanation

While creating or replacing the default object <*insert_4*> for IBM WebSphere MQ queue manager <*insert_5*> an error occurred. The error was due to an unexpected error, error code <*insert_1*> (X<*insert_2*>).

Response

Check this log for more details of what the problem may be. Make sure there is sufficient resources such as disk space and storage. For damaged or corrupted objects, replace these from backup objects. If all else fails, delete the queue manager <*insert_5*> using dltmqm and create it again using crtmqm.

AMQ8060

IBM WebSphere MQ queue manager <*insert_5*> started as a standby.

Severity

0 : Information

Explanation

Queue manager <*insert_5*> started as a standby instance, ready to become the primary instance if the existing primary instance fails.

Response

None.

AMQ8061 (Windows)

Command <*insert_4*> is not valid.

Severity

10 : Warning

Explanation

The command <*insert_4*> at line <*insert_1*> in the IBM WebSphere MQ service command file <*insert_3*> for queue manager <*insert_5*> is not valid for use in the service command file. The line is ignored.

Response

Check the contents of the file and retry the operation.

AMQ8062 (Windows)

Unexpected return code, <*insert_1*>, from command <*insert_3*>.

Severity

10 : Warning

Explanation

An unexpected return code, <*insert_1*>, was returned by command <*insert_3*>. This command was issued by the IBM WebSphere MQ service for queue manager <*insert_4*>.

Response

Verify that the command and parameters are correct.

AMQ8063 (Windows)

Not authorized to issue command <*insert_3*>.

Severity

20 : Error

Explanation

The current user <*insert_5*> is not authorized to issue the command <*insert_3*>. This can occur if the user is a member of the Administrators group but is not currently elevated. The command is ignored.

Response

Add the user to the local 'mqm' security group and retry the operation.

AMQ8064 (Windows)

Not authorized to start trusted application.

Severity

20 : Error

Explanation

The user <*insert_5*> is not authorized to start the trusted application <*insert_3*>. The application has not started.

Response

Add the user to the local 'mqm' security group and restart the application.

AMQ8065 (Windows)

Local group <*insert_3*> not found.

Severity

20 : Error

Explanation

The local group <*insert_3*> is unavailable. It is not possible to verify that the user is authorized. The function cannot continue.

Response

Create the required local group and retry the operation.

AMQ8066 (Windows)

Local mqm group not found.

Severity
20 : Error

Explanation
The local mqm group is unavailable. It is not possible to verify that the user is authorized. The function cannot continue.

Response
Create the local mqm group and retry the operation.

AMQ8067
IBM WebSphere MQ channel auto-defined.

Severity
0 : Information

Explanation
Channel <*insert_5*> auto-defined.

Response
None.

AMQ8068
Setup completed.

Severity
0 : Information

Response
None.

AMQ8069
ApplicationGroup for the crtmqm command does not contain the mqm userid.

Severity
40 : Stop Error

Explanation
IBM WebSphere MQ queue manager <*insert_5*> not created. The ApplicationGroup specified for the crtmqm command must contain the mqm userid when the RestrictedMode option (-g) is specified.

Response
None.

AMQ8070
ApplicationGroup for crtmqm command is not defined.

Severity
40 : Stop Error

Explanation
IBM WebSphere MQ queue manager <*insert_5*> not created. RestrictedMode option (-g) specified, but the ApplicationGroup does not exist.

Response
None.

AMQ8071
RestrictedMode option not supported on this platform.

Severity
40 : Stop Error

Explanation
IBM WebSphere MQ queue manager <*insert_5*> not created. The RestrictedMode option was specified but is not supported on this platform.

Response
None.

AMQ8072 (Windows)
Not authorized to administer channels.

Severity
10 : Warning

Explanation
The command server for queue manager <*insert_3*> received an administration command for channels. The user <*insert_5*> is not authorized to administer IBM WebSphere MQ channels. The command server has not processed the command.

Response
Add the user to the local 'mqm' security group, and ensure that the security policy is set as required.

AMQ8073 (Windows)
Authorization failed because SID: (<*insert_3*>) could not be resolved.

Severity
10 : Warning

Explanation
The object authority manager was unable to resolve the specified SID into entity and domain information.

Response
Ensure that the application provides a SID that is recognized on this system, that all necessary domain controllers are available, and that the security policy is set as you required.

AMQ8074 (Windows)
Authorization failed as the SID <*insert_3*> does not match the entity <*insert_4*>.

Severity
10 : Warning

Explanation
The object authority manager received inconsistent data - the supplied SID does not match that of the supplied entity information.

Response
Ensure that the application is supplying valid entity and SID information.

AMQ8075 (Windows)
Authorization failed because the SID for entity <*insert_3*> cannot be obtained.

Severity
10 : Warning

Explanation
The object authority manager was unable to obtain a SID for the specified entity.

Response
Ensure that the entity is valid, and that all necessary domain controllers are available.

AMQ8076 (Windows)
Authorization failed because no SID was supplied for entity <*insert_3*>.

Severity
10 : Warning

Explanation
The object authority manager was not supplied with SID information for the specified entity, and the security policy is set to 'NTSIDsRequired'.

Response
Ensure that the application is supplying a valid SID, and that the security policy is set as you require.

AMQ8077
Entity <*insert_3*> has insufficient authority to access object <*insert_4*>.

Severity
10 : Warning

Explanation
The specified entity is not authorized to access the required object. The following requested permissions are unauthorized: <*insert_5*>

Response
Ensure that the correct level of authority has been set for this entity against the required object, or ensure that the entity is a member of a privileged group.

AMQ8078
Waiting for queue manager <*insert_3*> to end.

Severity
0 : Information

Response
None.

AMQ8079 (Windows)
Access was denied when attempting to retrieve group membership information for user <*insert_3*>.

Severity
10 : Warning

Explanation
IBM WebSphere MQ, running with the authority of user <*insert_4*>, was unable to retrieve group membership information for the specified user.

Response
Ensure Active Directory access permissions allow user <*insert_4*> to read group memberships for user <*insert_3*>. To retrieve group membership information for a domain user, MQ must run with the authority of a domain user and a domain controller must be available.

AMQ8079 (IBM i)
IBM WebSphere MQ trigger monitor job started.

Severity
0 : Information

Explanation
The message queue manager trigger monitor job has been started for queue manager <*insert_3*> to process messages on the selected initiation queue. See previously issued messages for job details.'

Response
None.

AMQ8080 (IBM i)
IBM WebSphere MQ trigger monitor job start failed.

Severity
40 : Stop Error

Explanation
Message queue manager trigger job failed to start for manager <*insert_3*>. Failure reason code is <*insert_2*>. See previously issued messages for more information.'

Response
None.

AMQ8081 (Windows)
Not authorized to administer queue managers.

Severity
10 : Warning

Explanation
The command server for queue manager <*insert_3*> received an administration command for a queue manager. The user <*insert_5*> is not authorized to administer IBM WebSphere MQ queue managers. The command server has not processed the command.

Response
Add the user to the local 'mqm' security group, and ensure that the security policy is set as required.

AMQ8082 (Windows)
Not authorized to administer clusters.

Severity
10 : Warning

Explanation
The command server for queue manager <*insert_3*> received an administration command for clusters. The user <*insert_5*> is not authorized to administer IBM WebSphere MQ clusters. The command server has not processed the command.

Response
Add the user to the local 'mqm' security group, and ensure that the security policy is set as required.

AMQ8083
IBM WebSphere MQ queue manager <*insert_3*> starting.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_3*> starting.

Response
None.

AMQ8084
IBM WebSphere MQ connection not found.

Severity
0 : Information

Explanation
The connection specified does not exist.

Response
Correct the connection name and then try the command again.

AMQ8085
IBM WebSphere MQ queue manager <*insert_3*> is being started for replay.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_3*> is being started for replay. The strmqm command has been issued with the '-r' option. see the IBM WebSphere MQ System Administration documentation for details.

Response
None.

AMQ8086
IBM WebSphere MQ queue manager <*insert_3*> is being activated.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_3*> is being activated. The strmqm command has been issued with the '-a' option. see the IBM WebSphere MQ System Administration documentation for details.

Response
None.

AMQ8086 (IBM i)
IBM WebSphere MQ queue manager <*insert_3*> is being activated.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_3*> is being activated. The STRMQM command has been issued with the ACTIVATE(*YES) option. see the IBM WebSphere MQ System Administration documentation for further details.

Response
None.

AMQ8087
Attempt to migrate listener <*insert_3*> to a QM object failed with <*insert_1*>.

Severity
20 : Error

Explanation
Whilst processing legacy services, listener <*insert_3*> could not be migrated to an MQ object named <*insert_4*>, the object creation failed with <*insert_1*>.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8088
Attempt to migrate trigger monitor <*insert_3*> to a QM object failed with <*insert_1*>.

Severity
20 : Error

Explanation
Whilst processing legacy services, trigger monitor <*insert_3*> could not be migrated to an MQ object named <*insert_4*>, the object creation failed with <*insert_1*>.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8089
Attempt to migrate channel service <*insert_3*> to a QM object failed with <*insert_1*>.

Severity
20 : Error

Explanation
Whilst processing legacy services, channel service <*insert_3*> could not be migrated to an MQ object named <*insert_4*>, the object creation failed with <*insert_1*>.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8090
Attempt to migrate channel initiator <*insert_3*> to a QM object failed with <*insert_1*>.

Severity
20 : Error

Explanation
Whilst processing legacy services, channel initiator <*insert_3*> could not be migrated to an MQ object named <*insert_4*>, the object creation failed with <*insert_1*>.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8091
Attempt to migrate custom service <*insert_3*> to a QM object failed with <*insert_1*>.

Severity
20 : Error

Explanation
Whilst processing legacy services, custom service <*insert_3*> could not be migrated to an MQ object named <*insert_4*>, the object creation failed with <*insert_1*>.

Response
Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8092
Service migration statistics : <*insert_1*> migrated. <*insert_2*> failed.

Severity
0 : Information

Explanation
Information on the number of service objects migrated from previous versions of IBM WebSphere MQ for Windows services as well as any failures that occurred.

Response
None.

AMQ8093
IBM WebSphere MQ subscription changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ subscription <*insert_3*> changed.

Response
None.

AMQ8094
IBM WebSphere MQ subscription created.

Severity
0 : Information

Explanation
IBM WebSphere MQ subscription <*insert_3*> created.

Response
None.

AMQ8095
IBM WebSphere MQ subscription deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ subscription <*insert_3*> deleted.

Response
None.

AMQ8096
IBM WebSphere MQ subscription inquired.

Severity
0 : Information

Explanation
IBM WebSphere MQ subscription <*insert_3*> inquired.

Response
None.

AMQ8097
Default object <*insert_3*>. Unable to change attribute <*insert_1*> to value <*insert_2*>.

Severity
20 : Error

Explanation
While migrating a queue manager to a newer release an attempt was made to change the value

of an attribute of one of the default objects. The attribute of the above named default object could not be changed. While modifying the integer attribute <*insert_1*> of the default object <*insert_3*> for IBM WebSphere MQ queue manager <*insert_4*> an unexpected error occurred.

Response
The most likely cause of this error is that object <*insert_3*> has been redefined to be an object of a conflicting type for which the attribute <*insert_1*> is not applicable. For example if a default queue which was originally a local queue is changed to be an alias queue then the queue manager could fail to set the attribute MQIA_MAX_MSG_LENGTH (13) as MAXMSGL is not an attribute supported by alias queues. Review the customer configuration to see if a corresponding change needs to be made to the customer defined replacement for the named default object.

AMQ8098
IBM WebSphere MQ subscription copied.

Severity
0 : Information

Explanation
IBM WebSphere MQ subscription <*insert_3*> copied.

Response
None.

AMQ8099
IBM WebSphere MQ subscription status inquired.

Severity
0 : Information

Explanation
IBM WebSphere MQ subscription status <*insert_3*> inquired.

Response
None.

AMQ8101
IBM WebSphere MQ error (<*insert_1*>) has occurred.

Severity
40 : Stop Error

Explanation
An unexpected reason code with hexadecimal value <*insert_1*> was received from the IBM WebSphere MQ queue manager during command processing. (Note that hexadecimal values in the range X'07D1'-X'0BB7' correspond to MQI reason codes 2001-2999.) More information might be available in the log. If the reason code value indicates that the error was associated with a particular parameter, the parameter concerned is <*insert_4*>.

Response
Correct the error and then try the command again.

AMQ8102
IBM WebSphere MQ object name specified in <*insert_4*> not valid.

Severity
30 : Severe error

Explanation
The object name <*insert_3*> specified in <*insert_4*> is not valid. The length of the name must not exceed 48 characters, or 20 characters if it is a channel name. The name should contain the following characters only: lowercase a-z, uppercase A-Z, numeric 0-9, period (.), forward slash (/), underscore (_) and percent sign (%).

Response
Change the length of the parameter value or change the parameter value to contain a valid combination of characters, then try the command again.

AMQ8103
Insufficient storage available.

Severity
40 : Stop Error

Explanation
There was insufficient storage available to perform the requested operation.

Response
Free some storage and then try the command again.

AMQ8104
IBM WebSphere MQ directory <*insert_3*> not found.

Severity
40 : Stop Error

Explanation
Directory <*insert_3*> was not found. This directory is created when IBM WebSphere MQ is installed successfully. Refer to the log for more information.

Response
Verify that installation of IBM WebSphere MQ was successful. Correct the error and then try the command again.

AMQ8105
Object error.

Severity
40 : Stop Error

Explanation
An object error occurred. Refer to the log for more information.

Response
Correct the error and then try the command again.

AMQ8106
IBM WebSphere MQ queue manager being created.

Severity
0 : Information

Explanation
The queue manager is being created.

Response
Wait for the creation process to complete and then try the command again.

AMQ8107
IBM WebSphere MQ queue manager running.

Severity
10 : Warning

Explanation
The queue manager is running.

Response
None.

AMQ8108
IBM WebSphere MQ queue manager <*insert_3*> ending.

Severity
10 : Warning

Explanation
The queue manager <*insert_3*> is ending.

Response
Wait for the queue manager to end and then try the command again.

AMQ8109
IBM WebSphere MQ queue manager being deleted.

Severity
0 : Information

Explanation
The queue manager is being deleted.

Response
Wait for the deletion process to complete.

AMQ8110
IBM WebSphere MQ queue manager already exists.

Severity
40 : Stop Error

Explanation
The queue manager <*insert_5*> already exists.

Response
None.

AMQ8112 (IBM i)
PRCNAME not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The PRCNAME parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the PRCNAME parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8113 (IBM i)
TRGENBL not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The TRGENBL parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the TRGENBL parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8114 (IBM i)
GETENBL not allowed with queue type *RMT.

Severity
40 : Stop Error

Explanation
The GETENBL parameter may not be specified for a queue of type *RMT.

Response
Remove the GETENBL parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8115 (IBM i)
SHARE not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The SHARE parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the SHARE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8116 (IBM i)
MSGDLYSEQ not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The MSGDLYSEQ parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the MSGDLYSEQ parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8117
IBM WebSphere MQ queue manager deletion incomplete.

Severity
40 : Stop Error

Explanation
Deletion of queue manager <*insert_5*> was only partially successful. An object was not found, or could not be deleted. Refer to the log for more information.

Response
Delete any remaining queue manager objects.

AMQ8118
IBM WebSphere MQ queue manager does not exist.

Severity
40 : Stop Error

Explanation
The queue manager <*insert_5*> does not exist.

Response
Either create the queue manager (crtmqm command) or correct the queue manager name used in the command and then try the command again.

AMQ8119
Unsupported threading model detected.

Severity
20 : Error

Explanation
The command executed could not run because the current threading model does not contain the required level of functionality.

Response
On Linux this may be caused by using a threading model such as LinuxThreads which does not provide process-shared mutex support. On some systems, the setting of the environment variable LD_ASSUME_KERNEL causes LinuxThreads to be used instead of native kernel threads.

AMQ8119 (IBM i)
TRGTYPE not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The TRGTYPE parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the TRGTYPE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8120 (IBM i)
TRGDEPTH not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The TRGDEPTH parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the TRGDEPTH parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8121 (IBM i)
TRGMSGPTY not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The TRGMSGPTY parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the TRGMSGPTY parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8122 (IBM i)
TRGDATA not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The TRGDATA parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the TRGDATA parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8123 (IBM i)
RTNITV not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The RTNITV parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the RTNITV parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8124 (IBM i)
MAXMSGLEN not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The MAXMSGLEN parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the MAXMSGLEN parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8125 (IBM i)
BKTTHLD not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The BKTTHLD parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the BKTTHLD parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8126 (IBM i)
BKTQNAME not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The BKTQNAME parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the BKTQNAME parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8127 (IBM i)
INITQNAME not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The INITQNAME parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the INITQNAME parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8128 (IBM i)
USAGE not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The USAGE parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the USAGE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8129 (IBM i)
DFNTYPE only allowed with queue type *MDL.

Severity
40 : Stop Error

Explanation
The DFNTYPE parameter may only be specified for a queue of type *MDL.

Response
Remove the DFNTYPE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8130 (IBM i)
TGTQNAME only allowed with queue type *ALS.

Severity
40 : Stop Error

Explanation
The TGTQNAME parameter may only be specified for a queue of type *ALS.

Response
Remove the TGTQNAME parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8131 (IBM i)
RMTQNAME only allowed with queue type *RMT.

Severity
40 : Stop Error

Explanation
The RMTQNAME parameter may only be specified for a queue of type *RMT.

Response
Remove the RMTQNAME parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8132 (IBM i)
RMTMQMNAME only allowed with queue type *RMT.

Severity
40 : Stop Error

Explanation
The RMTMQMNAME parameter may only be specified for a queue of type *RMT.

Response
Remove the RMTMQMNAME parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8133 (IBM i)
TMQNAME only allowed with queue type *RMT.

Severity
40 : Stop Error

Explanation
The TMQNAME parameter may only be specified for a queue of type *RMT.

Response
Remove the TMQNAME parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8134 (IBM i)
HDNBKTCNT not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The HDNBKTCNT parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the HDNBKTCNT parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8135
Not authorized.

Severity
40 : Stop Error

Explanation
You are not authorized to perform the requested operation for the IBM WebSphere MQ object. Either you are not authorized to perform the requested operation, or you are not authorized to the specified MQ object. For a copy command, you may not be authorized to the specified source MQ object, or, for a create command, you may not be authorized to the system default MQ object of the specified type. If creating or altering a subscription it may also indicate that the subscribing user does not exist or have the required authority to the destination queue.

Response
Obtain the necessary authority from your security officer or IBM WebSphere MQ administrator. Then try the command again. If you are running amqmdain on the Windows platform, the user MUSR_MQADMIN may not be authorized.

AMQ8136 (IBM i)
Error detected by prompt control program.

Severity
30 : Severe error

Explanation
A prompt control program detected errors.

Response
See the previously listed messages in the job log. Correct the errors and then prompt for the command again.

AMQ8137
IBM WebSphere MQ queue manager already starting.

Severity
40 : Stop Error

Explanation
The strmqm command was unsuccessful because the queue manager <*insert_5*> is already starting.

Response
Wait for the strmqm command to complete.

AMQ8138
The IBM WebSphere MQ queue has an incorrect type.

Severity
40 : Stop Error

Explanation
The operation is not valid with queue <*insert_5*> because it is not a local queue.

Response
Change the QNAME parameter to specify a queue of the correct type.

AMQ8139
Already connected.

Severity
20 : Error

Explanation
A connection to the IBM WebSphere MQ queue manager already exists.

Response
None.

AMQ8140
Resource timeout error.

Severity
40 : Stop Error

Explanation
A timeout occurred in the communication between internal WebSphere MQ queue manager components. This is most likely to occur when the system is heavily loaded.

Response
Wait until the system is less heavily loaded, then try the command again.

AMQ8141
IBM WebSphere MQ queue manager starting.

Severity
40 : Stop Error

Explanation
The queue manager <*insert_5*> is starting.

Response
Wait for the queue manager startup process to complete and then try the command again.

AMQ8142
IBM WebSphere MQ queue manager stopped.

Severity
40 : Stop Error

Explanation
The queue manager <*insert_5*> is stopped.

Response
Use the strmqm command to start the queue manager, and then try the command again.

AMQ8143
IBM WebSphere MQ queue not empty.

Severity
40 : Stop Error

Explanation
The queue <*insert_5*> specified in <*insert_2*> is not empty or contains uncommitted updates.

Response
Commit or roll back any uncommitted updates. If the command is DELETE QLOCAL, use the CLEAR QLOCAL command to clear the messages from the queue. Then try the command again.

AMQ8144
Log not available.

Severity
40 : Stop Error

Explanation
The IBM WebSphere MQ logging resource is not available.

Response
Use the dltmqm command to delete the queue manager and then the crtmqm command to create the queue manager. Then try the command again.

AMQ8145
Connection broken.

Severity
40 : Stop Error

Explanation
The connection to the IBM WebSphere MQ queue manager failed during command processing. This may be caused by an endmqm command being issued by another user, or by a queue manager error.

Response
Use the strmqm command to start the message queue manager, wait until the message queue manager has started, and try the command again.

AMQ8146
IBM WebSphere MQ queue manager not available.

Severity
40 : Stop Error

Explanation
The queue manager is not available because it has been stopped or has not been created.

Response
Use the crtmqm command to create the message queue manager, or the strmqm command to start the message queue manager as necessary. Then try the command again.

AMQ8146 (IBM i)
IBM WebSphere MQ queue manager not available.

Severity
40 : Stop Error

Explanation
The queue manager is not available because it has been stopped or has not been created.

Response

Use the CRTMQM command to create the message queue manager or the STRMQM command to start the message queue manager as necessary, then retry the command. If a queue manager was not specified, ensure that a default queue manager has been created and is started using the WRKMQM command.

AMQ8147

IBM WebSphere MQ object <*insert_3*> not found.

Severity

40 : Stop Error

Explanation

If the command entered was Change or Display, the object <*insert_3*> specified does not exist. If the command entered was Copy, the source object does not exist. If the command entered was Create, the system default MQ object of the specified type does not exist.

Response

Correct the object name and then try the command again or, if you are creating a new queue or process object, either specify all parameters explicitly or ensure that the system default object of the required type exists. The system default queue names are SYSTEM.DEFAULT.LOCAL.QUEUE, SYSTEM.DEFAULT.ALIAS.QUEUE and SYSTEM.DEFAULT.REMOTE.QUEUE. The system default process name is SYSTEM.DEFAULT.PROCESS.

AMQ8147 (IBM i)

IBM WebSphere MQ object <*insert_3*> not found.

Severity

40 : Stop Error

Explanation

If the command entered was Change, Delete or Display, the MQ object <*insert_3*> specified does not exist. If the command entered was Copy, the source MQ object does not exist. If the command entered was Create, the system default MQ object of the specified type does not exist.

Response

Correct the MQ object name and then try the command again or, if you are creating a new MQ object, either specify all parameters explicitly or ensure that the system default object of the required type exists.

AMQ8148

IBM WebSphere MQ object in use.

Severity

40 : Stop Error

Explanation

The object <*insert_3*> is in use by an MQ application program.

Response

Wait until the object is no longer in use and then try the command again. If the command is ALTER or CHANGE, specify FORCE to force the processing of the object regardless of any application program affected by the change. If the object is the dead-letter queue and the open input count is nonzero, it may be in use by an MQ channel. If the object is another queue object with a nonzero open output count, it may be in use by a MQ channel (of type RCVR or RQSTR). In either case, use the STOP CHANNEL and START CHANNEL commands to stop and restart the channel in order to solve the problem. To alter the queue USAGE the FORCE option must be used if the queue is not empty.

AMQ8149

IBM WebSphere MQ object damaged.

Severity
40 : Stop Error

Explanation
The object <*insert_3*> specified in <*insert_4*> is damaged.

Response
The object contents are not valid. Issue the DISPLAY CHANNEL, DISPLAY QUEUE, or DISPLAY PROCESS command, as required, to determine the name of the damaged object. Issue the DEFINE command, for the appropriate object type, to replace the damaged object, then try the command again.

AMQ8150
IBM WebSphere MQ object already exists.

Severity
40 : Stop Error

Explanation
The object <*insert_3*> specified on the <*insert_5*> command could not be created because it already exists.

Response
Check that the name is correct and try the command again specifying REPLACE, or delete the object. Then try the command again.

AMQ8151
IBM WebSphere MQ object has different type.

Severity
40 : Stop Error

Explanation
The type specified for object <*insert_3*> is different from the type of the object being altered or defined.

Response
Use the correct MQ command for the object type, and then try the command again.

AMQ8152
Source IBM WebSphere MQ object has different type.

Severity
40 : Stop Error

Explanation
The type of the source object is different from that specified.

Response
Correct the name of the command, or source object name, and then try the command again, or try the command using the REPLACE option.

AMQ8153
Insufficient disk space for the specified queue.

Severity
40 : Stop Error

Explanation
The command failed because there was insufficient disk space available for the specified queue.

Response
Release some disk space and then try the command again.

AMQ8154
API exit load error.

Severity
40 : Stop Error

Explanation
The IBM WebSphere MQ queue manager was unable to load the API crossing exit.

Response
Ensure that the API crossing exit program is valid, and that its name and directory are correctly specified. Correct any error and then try the command again.

AMQ8155
Connection limit exceeded.

Severity
40 : Stop Error

Explanation
The queue manager connection limit has been exceeded.

Response
The maximum limit on the number of IBM WebSphere MQ application programs that may be connected to the queue manager has been exceeded. Try the command later.

AMQ8156
IBM WebSphere MQ queue manager quiescing.

Severity
40 : Stop Error

Explanation
The queue manager is quiescing.

Response
The queue manager was stopping with -c specified for endmqm. Wait until the queue manager has been restarted and then try the command again.

AMQ8157
Security error.

Severity
40 : Stop Error

Explanation
An error was reported by the security manager program.

Response
Inform your systems administrator, wait until the problem has been corrected, and then try the command again.

AMQ8158 (IBM i)
API exit not found.

Severity
40 : Stop Error

Explanation
The API crossing exit program was not found.

Response
Ensure that the API crossing exit program for the MQI exists, and that its name and library are correctly specified. Correct any errors and then try the command again.

AMQ8159 (IBM i)
MAXDEPTH not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The MAXDEPTH parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the MAXDEPTH parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8160 (IBM i)
DFTSHARE not allowed with queue type *ALS or *RMT.

Severity
40 : Stop Error

Explanation
The DFTSHARE parameter may not be specified for a queue of type *ALS or *RMT.

Response
Remove the DFTSHARE parameter from the command or, if the command is CRTMQMQ, specify a different value for QTYPE. Then try the command again.

AMQ8161 (IBM i)
AUT(*MQMPASSID) only allowed with OBJTYPE(*ADM).

Severity
40 : Stop Error

Explanation
AUT(*MQMPASSID) may only be specified with OBJTYPE(*ADM).

Response
Change the AUT parameter to specify another value and then try the command again.

AMQ8162 (IBM i)
AUT(*MQMPASSALL) only allowed with OBJTYPE(*ADM).

Severity
40 : Stop Error

Explanation
AUT(*MQMPASSALL) may only be specified with OBJTYPE(*ADM).

Response
Change the AUT parameter to specify another value and then try the command again.

AMQ8163 (IBM i)
AUT(*MQMSETID) only allowed with OBJTYPE(*ADM).

Severity
40 : Stop Error

Explanation
AUT(*MQMSETID) may only be specified with OBJTYPE(*ADM).

Response
Change the AUT parameter to specify another value and then try the command again.

AMQ8164 (IBM i)
AUT(*MQMSETALL) only allowed with OBJTYPE(*ADM).

Severity
40 : Stop Error

Explanation
AUT(*MQMSETALL) may only be specified with OBJTYPE(*ADM).

Response
Change the AUT parameter to specify another value and then try the command again.

AMQ8165 (IBM i)
AUT(*MQMALTUSR) only allowed with OBJTYPE(*ADM).

Severity
40 : Stop Error

Explanation
AUT(*MQMALTUSR) may only be specified with OBJTYPE(*ADM).

Response
Change the AUT parameter to specify another value and then try the command again.

AMQ8166 (IBM i)
IBM WebSphere MQ reference object not found.

Severity
40 : Stop Error

Explanation
The object specified by the REFOBJ and REFOBJTYPE parameters does not exist.

Response
Correct the reference object name and type, and then try the command again.

AMQ8167 (IBM i)
Referenced object name not valid.

Severity
30 : Severe error

Explanation
The referenced object name specified in REFOBJ is not valid. The length of the name must not exceed 48 characters and the name should contain the following characters only: lowercase a-z, uppercase A-Z, numeric 0-9, period (.), forward slash (/), underscore (_) and percent sign (%).

Response
Change the length of the parameter value or change the parameter value to contain a valid combination of characters. Then try the command again.

AMQ8168 (IBM i)
User profile name for parameter USER not found.

Severity
30 : Severe error

Explanation
The user profile name specified for parameter USER could not be found on the system, and is not the special value *PUBLIC.

Response
Correct the user profile name, or use the Create User Profile (CRTUSRPRF) command to create the user profile then try the request again.

AMQ8169 (IBM i)
Authorization list for parameter AUTL does not exist.

Severity
30 : Severe error

Explanation
The authorization list specified for parameter AUTL does not exist. It may have been destroyed.

Response
Either specify an authorization list that exists, or create the authorization list using the Create Authorization List (CRTAUTL) command. Try the request again.

AMQ8170 (IBM i)
REFOBJTYPE(*OBJTYPE) and OBJTYPE(*ALL) cannot be used together.

Severity
30 : Severe error

Explanation
REFOBJTYPE(*OBJTYPE) can be specified only with a specific object type.

Response
Change the REFOBJTYPE or OBJTYPE input value to a specific object type. Then try the Grant Authority (GRTMQMAUT) command again.

AMQ8171 (IBM i)
Authority of *AUTL is only allowed with USER(*PUBLIC).

Severity
30 : Severe error

Explanation
AUT(*AUTL) was specified on either the Grant Authority (GRTMQMAUT) command or the Revoke Authority (RVKMQMAUT) command with the USER parameter not set to *PUBLIC. Only the authority for *PUBLIC can be deferred to the authorization list.

Response
Change the AUT parameter to the authorities that are correct for the users or change the USER parameter to *PUBLIC. Then try the command again.

AMQ8172
Already disconnected.

Severity
10 : Warning

Explanation
The MQI reason code of 2018 was returned from the IBM WebSphere MQ queue manager in response to an MQDISC request issued during command processing.

Response
None.

AMQ8173
No processes to display.

Severity
0 : Information

Explanation
There are no matching processes defined on this system.

Response
Using the DEFINE PROCESS command to create a process.

AMQ8174
No queues to display.

Severity
0 : Information

Explanation
There are no matching queues defined on this system.

Response
Use the appropriate command to define a queue of the type that you require, that is, DEFINE QALIAS, DEFINE QLOCAL, DEFINE QMODEL, or DEFINE QREMOTE.

AMQ8175 (IBM i)
IBM WebSphere MQ trace has started.

Severity
0 : Information

Explanation
The trace has started successfully.

Response
None.

AMQ8176 (IBM i)
IBM WebSphere MQ trace has been written.

Severity
0 : Information

Explanation
The trace has been written successfully.

Response
None.

AMQ8177 (IBM i)
IBM WebSphere MQ trace has stopped.

Severity
0 : Information

Explanation
The trace has stopped.

Response
None.

AMQ8178 (IBM i)
IBM WebSphere MQ trace did not start.

Severity
40 : Stop Error

Explanation
The trace did not start successfully.

Response
None.

AMQ8179 (IBM i)
IBM WebSphere MQ trace output error.

Severity
40 : Stop Error

Explanation
The trace was not output successfully.

Response
None.

AMQ8180 (IBM i)
IBM WebSphere MQ trace end request failed.

Severity
40 : Stop Error

Explanation
Your request to end the trace was not successful.

Response
None.

AMQ8181 (IBM i)
No jobs to display.

Severity
10 : Warning

Explanation
There are no matching jobs running on this system.

Response
Specify another job name from the STRMQMSRV command.

AMQ8182 (IBM i)
IBM WebSphere MQ trace already off.

Severity
10 : Warning

Explanation
An attempt was made to set trace off, but the trace is not active.

Response
None.

AMQ8183 (IBM i)
IBM WebSphere MQ trace already running.

Severity
10 : Warning

Explanation
An attempt was made to start trace, but trace is already running.

Response
Either leave trace running as it is, or, if you want to change the trace settings, turn trace off and then turn it on again with appropriate settings.

AMQ8184 (IBM i)
Requested job cannot be found

Severity
10 : Warning

Explanation
The job specified cannot be found in the table that controls IBM WebSphere MQ for IBM i trace. As a result no trace action can be performed.

Response
Specify an appropriate job name.

AMQ8185
Operating system object already exists.

Severity
40 : Stop Error

Explanation
The IBM WebSphere MQ object cannot be created because an object that is not known to MQ already exists in the MQ directory with the name that should be used for the new object. Refer to the log for previous messages.

Response
Remove the non-MQ object from the MQ library, and try the command again.

AMQ8186
Image not available for IBM WebSphere MQ object <*insert_5*>.

Severity
40 : Stop Error

Explanation
The object <*insert_5*> type <*insert_3*> cannot be re-created because the image is not fully available in the logs that are currently online. Refer to earlier messages in the error log for information about the logs that need to be brought online for this object to be re-created.

Response
Bring the relevant logs online, and try the command again.

AMQ8187
IBM WebSphere MQ object <*insert_5*> is currently open.

Severity
40 : Stop Error

Explanation
The object <*insert_5*>, type <*insert_3*>, is currently in use, so the <*insert_1*> command cannot be issued against it. If a generic list was presented to the command, the command is still issued against the other objects in the list.

Response
Wait until the object is no longer in use, and try the command again.

AMQ8188
Insufficient authorization to IBM WebSphere MQ object <*insert_5*>.

Severity
40 : Stop Error

Explanation
You are not authorized to issue the <*insert_1*> command against the object <*insert_5*> type <*insert_3*>. If a generic list was presented to the command, the command is still issued against the other objects in the list.

Response
Obtain sufficient authorization for the object, and retry the command.

AMQ8189 (IBM i)
IBM WebSphere MQ object <*insert_3*> is damaged.

Severity
40 : Stop Error

Explanation
The object <*insert_3*> type <*insert_4*> is damaged and the <*insert_5*> command cannot be issued against it. If a generic list was presented to the command then the command is still issued against the other objects in the list.

Response
Issue the appropriate DEFINE command for the object, specifying REPLACE, and then try the command again.

AMQ8190
<*insert_3*> succeeded on <*insert_1*> objects and failed on <*insert_2*> objects.

Severity
40 : Stop Error

Explanation
An operation performed on a generic list of objects was not completely successful.

Response
Examine the log for details of the errors encountered, and take appropriate action.

AMQ8191
IBM WebSphere MQ command server is starting.

Severity
40 : Stop Error

Explanation
The command server is starting.

Response
Wait for the strmqcsv command to complete and then try the operation again.

AMQ8191 (IBM i)
IBM WebSphere MQ command server is starting.

Severity
40 : Stop Error

Explanation
The command server is starting.

Response
Wait for the STRMQMCSVR command to complete and then try the operation again.

AMQ8192
IBM WebSphere MQ command server already starting.

Severity
40 : Stop Error

Explanation
The request to start the command server was unsuccessful because the command server is already starting.

Response
Wait for the strmqcsv command to complete.

AMQ8192 (IBM i)
IBM WebSphere MQ command server already starting.

Severity
40 : Stop Error

Explanation
The request to start the command server was unsuccessful because the command server is already starting.

Response
Wait for the STRMQMCSVR command to complete.

AMQ8193
IBM WebSphere MQ command server is ending.

Severity
40 : Stop Error

Explanation
The command server is ending.

Response
Wait for the endmqcsv command to complete and then try the command again.

AMQ8193 (IBM i)
IBM WebSphere MQ command server is ending.

Severity
40 : Stop Error

Explanation
The command server is ending.

Response
Wait for the ENDMQMCSVR command to complete and then try the command again.

AMQ8194
IBM WebSphere MQ command server already ending.

Severity
40 : Stop Error

Explanation
The end command server request was unsuccessful because the command server is already ending.

Response
Wait for the endmqcsv command to complete.

AMQ8194 (IBM i)
IBM WebSphere MQ command server already ending.

Severity
40 : Stop Error

Explanation
The end command server request was unsuccessful because the command server is already ending.

Response
Wait for the ENDMQMCSVR command to complete.

AMQ8195
IBM WebSphere MQ command server already running.

Severity
40 : Stop Error

Explanation
The strmqcsv command was unsuccessful because the command server is already running.

Response
None.

AMQ8195 (IBM i)
IBM WebSphere MQ command server already running.

Severity
40 : Stop Error

Explanation
The STRMQMCSVR command was unsuccessful because the command server is already running.

Response
None.

AMQ8196
IBM WebSphere MQ command server already stopped.

Severity
40 : Stop Error

Explanation
The request to end the command server was unsuccessful because the command server is already stopped.

Response
None.

AMQ8197
Deleted IBM WebSphere MQ queue damaged.

Severity
20 : Error

Explanation
The deleted MQ queue <*insert_3*> was damaged, and any messages it contained have been lost.

Response
None.

AMQ8198 (IBM i)
Program <*insert_3*> called with incorrect number of parameters.

Severity
20 : Error

Explanation
The number of parameters passed in the call to program <*insert_3*> is not correct.

Response
Correct the calling program and then retry the operation.

AMQ8199 (IBM i)
Error in call identifier parameter passed to program QMQM.

Severity
20 : Error

Explanation
The call identifier, the first parameter passed to program QMQM, is not in the required packed decimal format, or its value is not supported. Permitted values of the call identifier are contained in the RPG copy file CMQR.

Response
Correct the calling program, and retry the call.

AMQ8200 (IBM i)
MODENAME only allowed with TRPTYPE(*LU62).

Severity
40 : Stop Error

Explanation
The MODENAME parameter may only be specified with TRPTYPE(*LU62).

Response
Remove the MODENAME parameter from the command or change the TRPTYPE parameter value to specify *LU62 and then try the command again.

AMQ8201 (IBM i)
TPGMNAME only allowed with TRPTYPE(*LU62).

Severity
40 : Stop Error

Explanation
The TPGMNAME parameter may only be specified with TRPTYPE(*LU62).

Response
Remove the TPGMNAME parameter from the command or change the TRPTYPE parameter value to specify *LU62. Then try the command again.

AMQ8202
TMQNAME only allowed with channel type *SDR or *SVR.

Severity
40 : Stop Error

Explanation
The TMQNAME parameter may only be specified with channel type *SDR or *SVR.

Response
Remove the TMQNAME parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR or *SVR. Then try the command again.

AMQ8203 (IBM i)
CONNAME only allowed with channel type *SDR, *SVR, *RQSTR, *CLUSSDR, *CLTCN and *CLUSRCVR

Severity
40 : Stop Error

Explanation
The CONNAME parameter may only be specified with channel type *SDR, *SVR, *RQSTR, *CLUSSDR, *CLTCN or *CLUSRCVR.

Response
Remove the CONNAME parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RQSTR, *CLUSSDR, *CLTCN or *CLUSRCVR. Then try the command again.

AMQ8204
MCANAME only allowed with channel type *SDR, *SVR, *RQSTR, *CLUSSDR or *CLUSRCVR

Severity
40 : Stop Error

Explanation
The MCANAME parameter may only be specified with channel type *SDR, *SVR, *RQSTR, *CLUSSDR or *CLUSRCVR.

Response

Remove the MCANAME parameter from the command, or if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RQSTR, *CLUSSDR or *CLUSRCVR. Then try the command again.

AMQ8205

DSCITV only allowed with channel type *CLUSSDR, *CLUSRCVR, *SDR or *SVR.

Severity

40 : Stop Error

Explanation

The DSCITV parameter may only be specified with channel type *CLUSSDR, *CLUSRCVR, *SDR or *SVR.

Response

Remove the DSCITV parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSSDR, *CLUSRCVR, *SDR or *SVR. Then try the command again.

AMQ8206

SHORTRTY only allowed with channel type *CLUSSDR, CLUSRCVR, *SDR or *SVR.

Severity

40 : Stop Error

Explanation

The SHORTRTY parameter may only be specified with channel type *CLUSSDR, *CLUSRCVR, *SDR or *SVR.

Response

Remove the SHORTRTY parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSSDR, *CLUSRCVR, *SDR or *SVR. Then try the command again.

AMQ8207

SHORTTMR only allowed with channel type *CLUSSDR, CLUSRCVR, *SDR or *SVR.

Severity

40 : Stop Error

Explanation

The SHORTTMR parameter may only be specified with channel type *CLUSSDR, *CLUSRCVR, *SDR or *SVR.

Response

Remove the SHORTTMR parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSSDR, CLUSRCVR, *SDR or *SVR. Then try the command again.

AMQ8208

LONGRTY only allowed with channel type *CLUSSDR, *CLUSRCVR, *SDR or *SVR.

Severity

40 : Stop Error

Explanation

The LONGRTY parameter may only be specified with channel type *CLUSSDR, *CLUSRCVR, *SDR or *SVR.

Response

Remove the LONGRTY parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSSDR, CLUSRCVR, *SDR or *SVR. Then try the command again.

AMQ8209
LONGTMR only allowed with channel type *CLUSSDR, *CLUSRCVR, *SDR or *SVR.

Severity
40 : Stop Error

Explanation
The LONGTMR parameter may only be specified with channel type *CLUSSDR, *CLUSRCVR, *SDR or *SVR.

Response
Remove the LONGTMR parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSSDR, *CLUSRCVR, *SDR or *SVR. Then try the command again.

AMQ8210
PUTAUT only allowed with channel type *RCVR, *RQSTR or *CLUSRCVR

Severity
40 : Stop Error

Explanation
The PUTAUT parameter may only be specified with channel type *RCVR, *RQSTR or *CLUSRCVR.

Response
Remove the PUTAUT parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *RCVR, *RQSTR or *CLUSRCVR. Then try the command again.

AMQ8211
BATCHINT only allowed with channel type *SDR or *SVR.

Severity
40 : Stop Error

Explanation
The BATCHINT parameter may only be specified with channel type *SDR or *SVR.

Response
Remove the BATCHINT parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR or *SVR. Then try the command again.

AMQ8212 (IBM i)
TPGMNAME parameter required with TRPTYPE(*LU62).

Severity
40 : Stop Error

Explanation
A required parameter was not specified.

Response
Enter a value for parameter TPGMNAME.

AMQ8213 (IBM i)
TMQNAME parameter required with channel type *SDR or *SVR.

Severity
40 : Stop Error

Explanation
The TMQNAME parameter must be specified with channel type *SDR or *SVR.

Response
Enter a value for parameter TMQNAME.

AMQ8214
CONNAME parameter missing.

Severity
40 : Stop Error

Explanation
The CONNAME parameter must be specified with channel types SDR, RQSTR, CLNTCONN, and CLUSSDR. It is also required with channel type CLUSRCVR if the TRPTYPE is not TCP.

Response
Enter a value for parameter CONNAME.

AMQ8214 (IBM i)
CONNAME parameter missing.

Severity
40 : Stop Error

Explanation
The CONNAME parameter must be specified with channel types *SDR, *RQSTR, *CLTCN and *CLUSSDR. It is also required with channel type *CLUSRCVR if the TRPTYPE is not *TCP.

Response
Enter a value for parameter CONNAME.

AMQ8215 (IBM i)
CVTMSG only allowed with channel type *SDR, *SVR, *CLUSSDR or *CLUSRCVR.

Severity
40 : Stop Error

Explanation
The CVTMSG parameter may only be specified with channel type *SDR, *SVR, *CLUSSDR or *CLUSRCVR.

Response
Remove the CVTMSG parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *CLUSSDR or CLUSRCVR. Then try the command again.

AMQ8216 (IBM i)
MODENAME only allowed with TRPTYPE(*LU62).

Severity
40 : Stop Error

Explanation
The MODENAME parameter may only be specified with TRPTYPE(*LU62).

Response
Remove the MODENAME parameter from the command or change the TRPTYPE parameter value to specify *LU62. Then try the command again.

AMQ8217 (IBM i)
CONNAME only allowed with channel type *SDR, *SVR, *RQSTR, *CLUSSDR or CLUSRCVR.

Severity
40 : Stop Error

Explanation
The CONNAME parameter may only be specified with channel type *SDR, *SVR, *RQSTR, CLUSSDR or CLUSRCVR.

Response
Remove the CONNAME parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RQSTR, CLUSSDR or CLUSRCVR. Then try the command again.

AMQ8218
The system cannot accept the combination of parameters entered.

Severity
30 : Severe error

AMQ8219
Command server queue is open, retry later.

Severity
30 : Severe error

Response
Wait and try again later.

AMQ8220 (IBM i)
The PNGMQMCHL command has completed.

Severity
0 : Information

Explanation
The PNGMQMCHL command sent <*insert_1*> bytes of data to <*insert_3*> and received the data back in <*insert_4*>.<*insert_5*> seconds. The number of bytes will be less than the amount requested on the command, when the length requested is greater than the allowed maximum, in one communications transmission, for the operating system and communications protocol.

Response
None.

AMQ8221 (IBM i)
Ping data length truncated, specified length <*insert_1*>, actual length <*insert_2*>.

Severity
10 : Warning

Explanation
The length of the ping data sent was reduced because of constraints in the current configuration.

Response
None.

AMQ8222 (IBM i)
The data sent and received by the PNGMQMCHL command was not identical.

Severity
40 : Stop Error

Explanation
Ping data compare failed at offset <*insert_1*>, data sent <*insert_3*>, data received <*insert_4*>.

Response
This is probably due to a communications failure. Other messages may have been issued.

AMQ8223 (IBM i)
No channels to display.

Severity
0 : Information

Explanation
There are no channels defined on this system.

Response
Create a channel using the CRTMQMCHL command.

AMQ8224 (IBM i)
From channel <*insert_3*> not found.

Severity
30 : Severe error

Explanation
The source IBM WebSphere MQ channel does not exist.

Response
Correct the MQ channel name and then try the command again.

AMQ8225 (IBM i)
From channel and to channel names are equal.

Severity
30 : Severe error

Explanation
The same name has been specified for the from channel name and the to channel name.

Response
Choose two different names, of which the from channel must exist.

AMQ8226
IBM WebSphere MQ channel already exists.

Severity
40 : Stop Error

Explanation
The channel <*insert_3*> cannot be created because it already exists.

Response
Check that the name is correct and try the command again specifying REPLACE, or delete the channel and then try the command again.

AMQ8227
Channel <*insert_3*> not found.

Severity
30 : Severe error

Explanation
The channel could not be found.

Response
Correct the Channel Name if wrong and then try the command again. For DEFINE CHANNEL check that the Channel Name in error exists.

AMQ8229 (IBM i)
No message queue managers to display.

Severity
0 : Information

Explanation
There are no message queue managers to administer.

Response
Add a queue manager using PF6 or the ADMQMNAM command.

AMQ8230 (IBM i)
No queue manager objects to display.

Severity
0 : Information

Explanation
Either the queue manager has no objects to display (this is unlikely), or the selection criteria resulted in zero objects to display.

Response
Change or remove the selection criteria.

AMQ8231 (IBM i)
No responses to display.

Severity
0 : Information

Explanation
There are no commands or command responses to display.

Response
None.

AMQ8232 (IBM i)
No messages to display.

Severity
0 : Information

Explanation
The queue is empty, or the queue does not exist.

Response
None.

AMQ8233 (IBM i)
No message data to display.

Severity
0 : Information

Explanation
The message contains no data.

Response
None.

AMQ8234 (IBM i)
No response data to display.

Severity
0 : Information

Explanation
There is no response data to display for this command. This is probably because the command has not yet completed.

Response
None.

AMQ8235 (IBM i)
No command parameters to display.

Severity
0 : Information

Explanation
Some commands have no required parameters.

Response
None.

AMQ8236 (IBM i)
Channel <*insert_3*> not found.

Severity
30 : Severe error

Explanation
CHGMQMCHL was issued for a non-existent channel.

Response
Correct the IBM WebSphere MQ channel name and then try the command again.

AMQ8237 (IBM i)
NPMSPEED only allowed with channel type *SDR, *SVR, *RCVR *RQSTR, CLUSSDR or CLUSRCVR.

Severity
40 : Stop Error

Explanation
The NPMSPEED parameter may only be specified with channel type *SDR, *SVR, *RCVR *RQSTR, CLUSSDR or CLUSRCVR.

Response
Remove the NPMSPEED parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RCVR *RQSTR, CLUSSDR or CLUSRCVR. Then try the command again.

AMQ8238 (IBM i)
Queue manager connection already open.

Severity
30 : Severe error

Explanation
An MQCONN call was issued, but the thread or process is already connected to a different queue manager. The thread or process can connect to only one queue manager at a time.

Response
Use the MQDISC call to disconnect from the queue manager which is already connected, and then issue the MQCONN call to connect to the new queue manager. Disconnecting from the existing queue manager will close any queues which are currently open, it is recommended that any uncommitted units of work should be committed or backed out before the MQDISC call is used.

AMQ8239 (IBM i)
LOCLADDR not valid for channel type *RCVR or *SVRCN.

Severity
40 : Stop Error

Explanation
The LOCLADDR parameter may only be specified with channel type *SDR, *SVR, *RQSTR, *CLUSSDR, *CLUSRCVR or *CLTCN.

Response
Remove the CONNAME parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RQSTR, *CLUSSDR, *CLUSRCVR or *CLTCN. Then try the command again.

AMQ8240 (IBM i)
Unexpected error <*insert_1*> in <*insert_3*>.

Severity
40 : Stop Error

Explanation
The unexpected return code <*insert_1*> was returned during <*insert_3*> processing.

Response
This message is associated with an internal error. Use WRKPRB to record the problem identifier, and to save the QPSRVDMP, QPJOBLOG, and QPDSPJOB files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8241 (IBM i)
Unexpected message format <*insert_3*> received.

Severity
40 : Stop Error

Explanation
The unexpected message format <*insert_3*> was received in message on the internal reply queue.

Response
This message is probably a message sent erroneously to this queue. The message in error is written to the SYSTEM.ADMIN.EXCEPTION.QUEUE, where it may be viewed using the WRKMQMMSG command.

AMQ8242
SSLCIPH definition wrong.

Severity
40 : Stop Error

Explanation
The definition of the SSLCIPH parameter was wrong.

Response
Correct the SSLCIPH definition and try the command again.

AMQ8243
SSLPEER definition wrong.

Severity
40 : Stop Error

Explanation
The definition of the SSLPEER parameter was wrong. Possible causes may be that the syntax was invalid or that it contained an invalid attribute type.

Response
Correct the SSLPEER definition and try the command again.

AMQ8266 (IBM i)
No objects to display.

Severity
0 : Information

Explanation
There are no objects with the specified name and type.

Response
None.

AMQ8276
Display Connection details.

Severity
0 : Information

Explanation
The DISPLAY CONN command completed successfully. Details follow this message.

AMQ8278 (IBM i)
Maximum handle limit reached.

Severity
40 : Stop Error

Explanation
An attempt was made to exceed the maximum handle limit specified for the message queue manager.

Response
Increase the maximum handle limit specified for the message queue manager using the CHGMQM command. Then try the command again.

AMQ8280 (IBM i)
Queue does not exist.

Severity
30 : Severe error

Explanation
The queue being displayed does not exist on this queue manager.

Response
Check the name of the queue and retry the operation. If you are attempting to display a queue of type *ALS, check the queue definition references an existing queue definition.

AMQ8282 (IBM i)
Queue manager <*insert_3*> is not defined on the connected queue manager.

Severity
30 : Severe error

Explanation
Either the necessary queue manager name has been entered incorrectly on the add queue manager panel, or the queue manager has not been defined on the connected queue manager.

Response
Correct the name, or define <*insert_3*> on the connected queue manager by creating a local queue with name <*insert_3*> and usage *TMQ (transmission queue), and then creating sender and receiver channels on both the connected queue manager and queue manager <*insert_3*>.

AMQ8284 (IBM i)
This user is not authorized to queue <*insert_3*>.

Severity
40 : Stop Error

Explanation
Queue <*insert_3*> (queue manager <*insert_4*>) has not been authorized for your use.

Response
Have queue <*insert_3*> authorized for your use. If queue manager <*insert_4*> is not the local queue manager, you might not be authorized to the transmission queue for this queue manager.

AMQ8287
No channels with status to display.

Severity
0 : Information

Explanation
There are no channels having status information to display. This indicates either, that the channel has not been started previously, or, that the channel has been started but has not yet completed a transmission sequence.

Response
None.

AMQ8288 (IBM i)
Not authorized to command <*insert_1*>

Severity
40 : Stop Error

Explanation
You are not authorized to perform the requested operation for IBM WebSphere MQ command <*insert_1*>.

Response
Obtain the necessary authority from your IBM WebSphere MQ administrator. Then try the command again.

AMQ8289 (IBM i)
You are not authorized to the IBM WebSphere MQ command.

Severity
40 : Stop Error

Explanation
You are not authorized to the IBM WebSphere MQ command because your user profile is not a member of the QMQMADM group.

Response
Ask your MQ administrator to give your user profile *ALLOBJ authority, or add your user profile to the QMQMADM group (either as a primary or supplemental group)

AMQ8291 (IBM i)
IBM WebSphere MQ remote trace already running.

Severity
10 : Warning

Explanation
An attempt was made to start remote trace, but it is already running.

Response
Either leave remote trace running as it is, or, if you want to change the settings, turn remote trace off and then turn it on again with appropriate settings.

AMQ8294 (IBM i)
IBM WebSphere MQ remote trace already off.

Severity
10 : Warning

Explanation
An attempt was made to end remote trace, but it is already off.

Response
Leave remote trace off.

AMQ8295 (IBM i)
IBM WebSphere MQ object not secured by authorization list.

Severity
40 : Stop Error

Explanation
The specified object is not secured by the authorization list to be revoked from it.

Response
Use the display authority (DSPMQMAUT) command to determine what authorization list is securing the object, if any. Issue the RVKMQMAUT command again with the authorization list that is securing the the object to revoke the authorization list's authority.

AMQ8296
<*insert_1*> MQSC commands completed successfully.

Severity
0 : Information

Explanation
The <*insert_3*> command has completed successfully. The <*insert_1*> MQ commands from <*insert_5*> have been processed without error and a report written to the printer spool file.

Response
None.

AMQ8297
<*insert_1*> MQSC commands verified successfully.

Severity
0 : Information

Explanation
The <*insert_3*> command completed successfully. The <*insert_1*> MQ commands from <*insert_5*> have been verified and a report written to the printer spool file.

Response
None.

AMQ8298
Error report generated for MQSC command process.

Severity
40 : Stop Error

Explanation
The <*insert_5*> command attempted to process a sequence of MQ commands and encountered some errors, however, the operation may have partially completed.

Response

If the <*insert_5*> command was executed a report has been written to a printer spool file. Examine the spooled printer file for details of the errors encountered and correct the MQSC source in <*insert_3*> and retry the operation.

AMQ8299

Cannot open <*insert_3*> for MQSC process.

Severity

40 : Stop Error

Explanation

The <*insert_5*> command failed to open <*insert_3*> for MQ command processing.

Response

Check that the intended file exists, and has been specified correctly. Correct the specification or create the object, and try the operation again.

AMQ8300 (IBM i)

Too many exit programs/user data fields defined.

Severity

30 : Severe error

Explanation

An attempt was made to create or change a channel which had more than the allowed maximum of a total of six exit programs, user data fields, or both defined.

Response

Define the channel again so that a maximum of six exit programs, user data fields, or both are defined.

AMQ8301 (IBM i)

IBM WebSphere MQ storage monitor job could not be started.

Severity

50 : System Error

Explanation

An attempt to start the storage monitor process (job QMQM in subsystem QSYSWRK) was unsuccessful.

Response

Check the job log for the reason for the failure, and try the command again.

AMQ8302

Internal failure initializing IBM WebSphere MQ services.

Severity

50 : System Error

Explanation

An error occurred while attempting to initialize IBM WebSphere MQ services.

Response

A call to xcsInitialize ended with the FAIL, STOP, or STOP_ALL return code. Refer to the log for messages diagnosing this problem.

AMQ8303

Insufficient storage available to process request.

Severity

50 : System Error

AMQ8304
Tracing cannot be started. Too many traces are already running.

Severity
40 : Stop Error

Explanation
A maximum of 15 traces may be running concurrently. This number is already running.

Response
Stop one or more of the other traces and try the command again.

AMQ8305
Tracing cannot be started. Too many traces are already running.

Severity
40 : Stop Error

Explanation
A maximum of 9 traces can be running concurrently, and this number of traces is already running.

Response
Stop one or more of the other traces and try the command again.

AMQ8306 (IBM i)
BATCHSIZE only allowed with channel type *SDR, *SVR, *RCVR, *RQSTR, CLUSSDR or CLUSRCVR.

Severity
40 : Stop Error

Explanation
The BATCHSIZE parameter may only be specified with channel type *SDR, *SVR, *RCVR, *RQSTR, CLUSSDR or CLUSRCVR.

Response
Remove the BATCHSIZE parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RCVR *RQSTR, CLUSSDR or CLUSRCVR. Then try the command again.

AMQ8307 (IBM i)
SEQNUMWRAP only allowed with channel type *SDR, *SVR, *RCVR , *RQSTR, CLUSSDR or CLUSRCVR.

Severity
40 : Stop Error

Explanation
The SEQNUMWRAP parameter may only be specified with channel type *SDR, *SVR, *RCVR, *RQSTR, CLUSSDR or CLUSRCVR.

Response
Remove the SEQNUMWRAP parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RCVR *RQSTR, CLUSSDR or CLUSRCVR. Then try the command again.

AMQ8308 (IBM i)
MSGRTYEXIT only allowed with channel type *CLUSRCVR, *RCVR or *RQSTR.

Severity
40 : Stop Error

Explanation
The MSGRTYEXIT parameter may only be specified with channel type *CLUSRCVR, *RCVR or *RQSTR.

Response
Remove the MSGRTYEXIT parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSRCVR, *RCVR or *RQSTR. Then try the command again.

AMQ8309 (IBM i)
MSGRTYDATA only allowed with channel type *CLUSRCVR, *RCVR or *RQSTR.

Severity
40 : Stop Error

Explanation
The MSGRTYDATA parameter may only be specified with channel type *CLUSRCVR, *RCVR or *RQSTR.

Response
Remove the MSGRTYDATA parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSRCVR, *RCVR or *RQSTR. Then try the command again.

AMQ8310 (IBM i)
MSGRTYNBR only allowed with channel type *CLUSRCVR, *RCVR or *RQSTR.

Severity
40 : Stop Error

Explanation
The MSGRTYNBR parameter may only be specified with channel type *CLUSRCVR, *RCVR or *RQSTR.

Response
Remove the MSGRTYNBR parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSRCVR, *RCVR or *RQSTR. Then try the command again.

AMQ8311 (IBM i)
MSGRTYITV only allowed with channel type *CLUSRCVR, *RCVR or *RQSTR.

Severity
40 : Stop Error

Explanation
The MSGRTYITV parameter may only be specified with channel type *CLUSRCVR, *RCVR or *RQSTR.

Response
Remove the MSGRTYITV parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSRCVR, *RCVR or *RQSTR. Then try the command again.

AMQ8312 (IBM i)
CLUSTER only allowed with queue type *ALS, *LCL and *RMT.

Severity
40 : Stop Error

Explanation
The CLUSTER parameter may only be specified with queue type *ALS, *LCL and *RMT.

Response
Remove the CLUSTER parameter from the command or, if the command is CRTMQMQ, change the QTYPE parameter value to specify *ALS, *LCL or *RMT. Then try the command again.

AMQ8313 (IBM i)
CLUSNL only allowed with queue type *ALS, *LCL and *RMT.

Severity
40 : Stop Error

Explanation
The CLUSNL parameter may only be specified with queue type *ALS, *LCL and *RMT.

Response
Remove the CLUSNL parameter from the command or, if the command is CRTMQMQ, change the QTYPE parameter value to specify *ALS, *LCL or *RMT. Then try the command again.

AMQ8314 (IBM i)
DEFBIND only allowed with queue type *ALS, *LCL and *RMT.

Severity
40 : Stop Error

Explanation
The DEFBIND parameter may only be specified with queue type *ALS, *LCL and *RMT.

Response
Remove the DEFBIND parameter from the command or, if the command is CRTMQMQ, change the QTYPE parameter value to specify *ALS, *LCL or *RMT. Then try the command again.

AMQ8315
No namelists to display.

Severity
0 : Information

Explanation
There are no matching namelists defined on this system.

Response
Use the Create Namelist (CRTMQMNL) command to create a namelist.

AMQ8316
No cluster queue managers to display.

Severity
0 : Information

Explanation
There are no matching cluster queue managers defined on this system.

Response
None.

AMQ8317 (IBM i)
CLUSTER only allowed with channel type *CLUSSDR and *CLUSRCVR.

Severity
40 : Stop Error

Explanation
The CLUSTER parameter may only be specified with channel type *CLUSSDR and *CLUSRCVR.

Response

Remove the CLUSTER parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *CLUSSDR or *CLUSRCVR. Then try the command again.

AMQ8318 (IBM i)

CLUSNL only allowed with channel type *CLUSSDR and *CLUSRCVR.

Severity

40 : Stop Error

Explanation

The CLUSNL parameter may only be specified with channel type *CLUSSDR and *CLUSRCVR.

Response

Remove the CLUSNL parameter from the command or, if the command is CRTMQMCHL, change the CHLQTYPE parameter value to specify *CLUSSDR or *CLUSRCVR. Then try the command again.

AMQ8319

MSGEXIT only allowed with channel type *SDR, *SVR, *RCVR *RQSTR, *CLUSSDR or *CLUSRCVR.

Severity

40 : Stop Error

Explanation

The MSGEXIT parameter may only be specified with channel type *SDR, *SVR, *RCVR, *RQSTR, *CLUSSDR, or *CLUSRCVR.

Response

Remove the MSGEXIT parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR or *SVR or *RCVR or *RQSTR or *CLUSSDR or *CLUSRCVR. Then try the command again.

AMQ8320 (IBM i)

MSGUSRDATA only allowed with channel type *SDR, *SVR, *RCVR *RQSTR, or *CLUSSDR or *CLUSRCVR.

Severity

40 : Stop Error

Explanation

The MSGUSRDATA parameter may only be specified with channel type *SDR, *SVR, *RCVR *RQSTR, *CLUSSDR or *CLUSRCVR.

Response

Remove the MSGUSRDATA parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR or *SVR or *RCVR or *RQSTR or *CLUSSDR or *CLUSRCVR. Then try the command again.

AMQ8321 (IBM i)

Process <*insert_3*> is still running.

Severity

0 : Information

AMQ8322 (IBM i)

TIMEOUT only allowed with ENDCCTJOB(*YES).

Severity

40 : Stop Error

Explanation

The TIMEOUT parameter may only be specified when connected jobs are being ended with the ENDCCTJOB option set to *YES.

Response

Remove the TIMEOUT parameter from the command or, if you want to fully quiesce the queue manager, change the ENDCCTJOB parameter to *YES. Then try the command again.

AMQ8323 (IBM i)

OPTION(*PREEMPT) must not be used with ENDCCTJOB(*YES).

Severity

40 : Stop Error

Explanation

When performing a pre-emptive shutdown of the queue manager the ENDCCTJOB(*YES) parameter is not allowed.

Response

Change the ENDCCTJOB(*YES) parameter to ENDCCTJOB(*NO) or, if you want to fully quiesce the queue manager without doing a pre-emptive shutdown, change the OPTION(*PREEMPT) parameter to another value. Then try the command again.

AMQ8324 (IBM i)

OPTION(*WAIT) not allowed with MQMNAME(*ALL).

Severity

40 : Stop Error

Explanation

The OPTION(*WAIT) parameter is not allowed when performing a shutdown of all queue managers.

Response

Remove the OPTION(*WAIT) parameter from the command or, specify individual queue manager names to shut down the queue managers one-by-one with the OPTION(*WAIT) parameter. Then try the command again.

AMQ8325 (IBM i)

MQMNAME(*ALL) is not allowed with ENDCCTJOB(*NO).

Severity

40 : Stop Error

Explanation

The MQMNAME(*ALL) parameter is only allowed when performing a full shutdown of the queue managers.

Response

Specify individual queue manager names to shut the queue managers down one-by-one or change the ENDCCTJOB parameter to *YES. Then try the command again.

AMQ8330

Running

Severity

0 : Information

AMQ8331

Ended normally

Severity

0 : Information

AMQ8332
Ended immediately

Severity
0 : Information

AMQ8333
Ended preemptively

Severity
0 : Information

AMQ8334
Ended unexpectedly

Severity
0 : Information

AMQ8335
Starting

Severity
0 : Information

AMQ8336
Quiescing

Severity
0 : Information

AMQ8337
Ending immediately

Severity
0 : Information

AMQ8338
Ending preemptively

Severity
0 : Information

AMQ8339
Being deleted

Severity
0 : Information

AMQ8340
Not available

Severity
0 : Information

AMQ8341
SUBPOOL(<*insert_3*>)<*insert_4*>PID(<*insert_1*>)

Severity
0 : Information

AMQ8342
No authorities to display.

Severity
0 : Information

Explanation
There are no authority records defined on this system, satisfying the input parameters.

Response
Use the appropriate input to list all the authorities defined on the system, or enter the command again with different input..

AMQ8343
Running as standby

Severity
0 : Information

AMQ8343 (IBM i)
The requested operation is not valid for user QMQMADM.

Severity
0 : Information

Explanation
You are not allowed to completely delete the authorities assigned to user QMQMADM, for a valid IBM WebSphere MQ object, with the authority *REMOVE or *NONE.

Response
Remove QMQMADM from the list of users to this command.

AMQ8344
Running elsewhere

Severity
0 : Information

AMQ8344 (IBM i)
The delete option is only valid for a generic profile name.

Severity
0 : Information

Explanation
The delete option, which will delete this authority profile by removing all the users from this authority profile, is not valid for an object name or the special value &class.

Response
To delete users from an object, work from the WRKMQMAUTD command.

AMQ8345 (IBM i)
BATCHHB not valid for channel type *RCVR, *RQSTR, *SVRCN or *CLTCN.

Severity
40 : Stop Error

Explanation
The BATCHHB parameter may only be specified with channel type *SDR, *SVR, *CLUSSDR, or *CLUSRCVR.

Response
Remove the BATCHHB parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *CLUSSDR or *CLUSRCVR. Then try the command again.

AMQ8346 (IBM i)
Parameter mismatch between QMNAME and QMID.

Severity
40 : Stop Error

Explanation
The Queue Manager Name for Removal (QMNAME) parameter is not *QMID and there is a value for the Queue Manager Identifier for Removal (QMID) parameter.

Response
A value for QMID is not allowed unless QMNAME is *QMID. Change the value specified on the QMNAME parameter or the value of the QMID parameter and then try the request again.

AMQ8347 (IBM i)
USERID not valid for channel type *RCVR, *SVRCN or *CLUSRCVR.

Severity
40 : Stop Error

Explanation
The USERID parameter may only be specified with channel type *SDR, *SVR, *RQSTR, *CLUSSDR, or *CLTCN.

Response
Remove the USERID parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RQSTR, *CLUSSDR, or *CLTCN. Then try the command again.

AMQ8348 (IBM i)
PASSWORD not valid for channel type *RCVR, *SVRCN or *CLUSRCVR.

Severity
40 : Stop Error

Explanation
The PASSWORD parameter may only be specified with channel type *SDR, *SVR, *RQSTR, *CLUSSDR, or *CLTCN.

Response
Remove the PASSWORD parameter from the command or, if the command is CRTMQMCHL, change the CHLTYPE parameter value to specify *SDR, *SVR, *RQSTR, *CLUSSDR, or *CLTCN. Then try the command again.

AMQ8349 (IBM i)
Authority changes to <*insert_3*> failed.

Severity
40 : Stop Error

Explanation
Authority changes to an object were requested but could not be made.

Response
Check the authorities that you are granting are relevant to the object type of <*insert_3*>.

AMQ8350
Usage: dspmqver [-p Components] [-f Fields] [-b] [-v]

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ8351
IBM WebSphere MQ Java environment has not been configured correctly.

Severity
20 : Error

Explanation
A command was issued that attempted to run a Java application. However either a working JRE (Java Runtime Environment) was not found or the IBM WebSphere MQ Java environment variables have not been set up. The command could not be run successfully.

Response
Ensure that you have a working JRE (Java Runtime Environment) and that the IBM WebSphere MQ Java environment variables have been set using the setjmsenv script. Retry the command.

AMQ8352
IBM WebSphere MQ queue manager <*insert_5*> becoming primary instance.

Severity
0 : Information

Explanation
Queue manager <*insert_5*> was running previously as a standby instance and is now becoming the primary instance.

Response
None.

AMQ8353
Quiesce request accepted. The queue manager will stop when all outstanding work is complete, permitting switchover to a standby queue manager.

Severity
0 : Information

Explanation
You have requested that the queue manager end when there is no more work for it. In the meantime, it will refuse new applications that attempt to start, although it allows those already running to complete their work. Once the queue manager has stopped, a switchover to a standby queue manager is permitted.

Response
None.

AMQ8354
IBM WebSphere MQ queue manager <*insert_5*> ended, permitting switchover to a standby queue manager.

Severity
0 : Information

Explanation
IBM WebSphere MQ queue manager <*insert_5*> ended. Once the queue manager has stopped, a switchover to a standby queue manager is permitted.

Response
None.

AMQ8355
IBM WebSphere MQ standby queue manager <*insert_5*> not permitted to become a primary instance.

Severity
20 : Error

Explanation
IBM WebSphere MQ standby queue manager <*insert_5*> obtained a lock on its data in the

file-system but was not permitted to become a primary instance. The most likely cause is that the queue manager was stopped without permitting a switchover.

Response
None.

AMQ8367
Active instance of IBM WebSphere MQ queue manager <*insert_3*> not ended.

Severity
20 : Error

Explanation
You tried to end the local instance of IBM WebSphere MQ queue manager <*insert_3*> using the '-x' option, which ends a standby instance. The local instance is not a standby instance.

Response
Issue the endmqm command without the '-x' option.

AMQ8368
Standby instance of IBM WebSphere MQ queue manager <*insert_3*> not ended.

Severity
20 : Error

Explanation
You tried to end the local instance of IBM WebSphere MQ queue manager <*insert_3*>. It is a standby instance so you must specify the '-x' option of endmqm.

Response
Issue the endmqm command with the '-x' option.

AMQ8370
Usage: runmqdnm -q Queue -a Assembly

[-m QueueManager] [-c ClassName] [-u Text] [-s Syncpoint]

[-n MaxThreads] [-t Timeout] [-b BackoutThreshold]

[-r BackoutQueue] [-p Context] [-d]

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ8371
<*insert_3*> is not a valid command line option.

Severity
40 : Stop Error

Explanation
The option <*insert_3*> was specified on the command line to the application. This option is not a valid command line options for the application.

Response
Check the usage information for the application and then retry.

AMQ8372
The required command line option <*insert_3*> is missing.

Severity
40 : Stop Error

Explanation
The application expects several mandatory command line options. One of these, <*insert_3*>, was not specified.

Response
Check the usage information for the application and ensure that all required parameters are specified then retry.

AMQ8373
Invalid value specified for command line option <*insert_3*> (<*insert_4*>).

Severity
40 : Stop Error

Explanation
The value specified for command line option <*insert_3*> (<*insert_4*>) is invalid.

Response
Check the usage information for the application and ensure that all options specify values in the valid range then retry.

AMQ8374
IBM WebSphere MQ queue manager <*insert_3*> does not exist.

Severity
40 : Stop Error

Explanation
The IBM WebSphere MQ queue manager <*insert_3*> does not exist.

Response
Either create the queue manager (crtmqm command) or correct the queue manager name used in the command and then try the command again.

AMQ8375
IBM WebSphere MQ queue manager <*insert_3*> not available.

Severity
40 : Stop Error

Explanation
The IBM WebSphere MQ queue manager <*insert_3*> is not available because it has been stopped or is otherwise not contactable.

Response
Use the strmqm command to start the message queue manager as necessary or correct any intermittent problems (eg. network connectivity) then try the command again.

AMQ8376
IBM WebSphere MQ queue <*insert_3*> not found.

Severity
40 : Stop Error

Explanation
The queue <*insert_3*> could not be found, it may not have been created.

Response
Ensure that the name of the queue specified is correct, queue names are case sensitive. If the queue is not created, use the runmqsc command to create it. Then try the command again.

AMQ8377

Unexpected error <*insert_1*> was received by the application.

Severity

40 : Stop Error

Explanation

The error <*insert_1*> was returned unexpectedly to the application.

Response

Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8378

Unexpected exception received from .NET Framework

<*insert_3*>

Severity

40 : Stop Error

Explanation

The application received an exception from the underlying .NET framework, information about the exception follows:

<*insert_4*>

Response

Examine the information contained within the exception to determine if it is possible to resolve locally.

If it is not possible to resolve the problem locally, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8379

Assembly <*insert_3*> could not be loaded

Severity

40 : Stop Error

Explanation

The IBM WebSphere MQ .NET Monitor attempted to load assembly <*insert_3*> but received an exception from the underlying .NET framework indicating that it could not be found. <*insert_4*>

Response

Check that the assembly does exist and is accessible to the user running the application then retry.

If the assembly should be available, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8380

No classes implementing IMQObjectTrigger found in <*insert_3*>.

Severity

40 : Stop Error

Explanation

The IBM WebSphere MQ .NET monitor was unable to identify any classes in referenced assembly <*insert_3*> which implement the IMQObjectTrigger interface.

Response

It is a requirement of the IBM WebSphere MQ .NET monitor that either a single class implementing the IMQObjectTrigger interface exists in the referenced assembly or that a class is identified in that assembly to execute. Either modify the assembly to include a single class implementing IMQObjectTrigger or specify a class name on the command line and retry.

AMQ8381

Too many classes implementing IMQObjectTrigger (<*insert_1*>) found in <*insert_3*>.

Severity

40 : Stop Error

Explanation

The IBM WebSphere MQ .NET monitor found <*insert_1*> classes in referenced assembly <*insert_3*> all of which implement the IMQObjectTrigger interface.

Response

It is a requirement of the IBM WebSphere MQ .NET monitor that either a single class implementing the IMQObjectTrigger interface exists in the referenced assembly or that a class is identified in that assembly to execute. Either modify the assembly to include a single class implementing IMQObjectTrigger or specify a class name on the command line and retry.

AMQ8382

A Message breaking the backout threshold (<*insert_1*>) was moved to <*insert_4*>

Severity

10 : Warning

Explanation

Whilst processing queue <*insert_3*> a message with a backout count that exceeded the specified backout threshold (<*insert_1*>) was successfully moved to <*insert_4*>

Response

The message moved to the backout queue has a backout count greater than the backout threshold specified (or picked up from the input queue BOTHRESH attribute). You should investigate the reason why this message was rolled back onto the input queue and resolve that issue. If backout processing is not required, modify the command line options and or queue definitions to achieve the required behaviour from the .NET monitor.

AMQ8383

A Message breaking the backout threshold (<*insert_1*>) could not be moved.

Severity

40 : Stop Error

Explanation

While processing queue <*insert_3*> a message with a backout count that exceeded the specified backout threshold (<*insert_1*>) was encountered however, it was not possible to move it to either a backout queue or the dead-letter queue.

Response

Because it was not possible to move the backed out message to another queue, it has been left on the input queue. As a result, the .NET monitor has ended.

It is possible that the backout queue or dead-letter queue are full or disabled for put - in this case, resolve this problem first.

If backout processing should have resulted in the message being placed on another queue, check the command line options, input queue definition and queue manager dead-letter queue attribute to ensure that they are correct, then retry.

AMQ8390
Usage: endmqdnm -q Queue [-m QueueManager]

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ8391
<*insert_3*> is not a valid command line option.

Severity
40 : Stop Error

Explanation
The option <*insert_3*> was specified on the command line to the application. This option is not one of the valid set of command line options.

Response
Check the usage information for the application and then retry.

AMQ8392
The required command line option <*insert_3*> is missing.

Severity
40 : Stop Error

Explanation
The application expects mandatory command line options. One of these, <*insert_3*>, was not specified.

Response
Check the usage information for the application and ensure that all required parameters are specified then retry.

AMQ8393
Invalid value specified for command line option <*insert_3*> (<*insert_4*>).

Severity
40 : Stop Error

Explanation
The value specified for command line option <*insert_3*> (<*insert_4*>) is invalid.

Response
Check the usage information for the application and ensure that all options specify values in the valid range then retry.

AMQ8394
IBM WebSphere MQ queue manager <*insert_3*> does not exist.

Severity
40 : Stop Error

Explanation
The IBM WebSphere MQ queue manager <*insert_3*> does not exist.

Response

Either create the queue manager (crtmqm command) or correct the queue manager name used in the command and then try the command again.

AMQ8395

IBM WebSphere MQ queue manager <*insert_3*> not available.

Severity

40 : Stop Error

Explanation

The IBM WebSphere MQ queue manager <*insert_3*> is not available because it has been stopped or is otherwise not contactable.

Response

Use the strmqm command to start the message queue manager as necessary or correct any intermittent problems (eg. network connectivity) then try the command again.

AMQ8396

IBM WebSphere MQ queue <*insert_3*> not found.

Severity

40 : Stop Error

Explanation

The queue <*insert_3*> could not be found, it may not have been created.

Response

Ensure that the name of the queue specified is correct, queue names are case sensitive. If the queue is not created, use the runmqsc command to create it. Then try the command again.

AMQ8397

Unexpected error <*insert_1*> was received by the application.

Severity

40 : Stop Error

Explanation

The error <*insert_1*> was returned unexpectedly to the application.

Response

Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8398

Unexpected exception received from .NET Framework

<*insert_3*>

Severity

40 : Stop Error

Explanation

The application received an exception from the underlying .NET framework, information about the exception follows:

<*insert_4*>

Response

Examine the information contained within the exception to determine if it is possible to resolve locally.

If it is not possible to resolve the problem locally, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8401
<*insert_1*> MQSC commands read.

Severity
0 : Information

Explanation
The MQSC script contains <*insert_1*> commands.

Response
None.

AMQ8402
<*insert_1*> commands have a syntax error.

Severity
0 : Information

Explanation
The MQSC script contains <*insert_1*> commands having a syntax error.

Response
None.

AMQ8403
<*insert_1*> valid MQSC commands could not be processed.

Severity
0 : Information

Explanation
The MQSC script contains <*insert_1*> commands that failed to process.

Response
None.

AMQ8404
Command failed.

Severity
0 : Information

Explanation
An MQSC command has been recognized, but cannot be processed.

Response
None.

AMQ8405
Syntax error detected at or near end of command segment below:-

Severity
0 : Information

Explanation
The MQSC script contains <*insert_1*> commands having a syntax error.

Response
None.

AMQ8406
Unexpected 'end of input' in MQSC.

Severity
0 : Information

Explanation
An MQSC command contains a continuation character, but the 'end of input' has been reached without completing the command.

Response
None.

AMQ8407
Display Process details.

Severity
0 : Information

Explanation
The MQSC DISPLAY PROCESS command completed successfully, and details follow this message.

Response
None.

AMQ8408
Display Queue Manager details.

Severity
0 : Information

Explanation
The MQSC DISPLAY QMGR command completed successfully, and details follow this message.

Response
None.

AMQ8409
Display Queue details.

Severity
0 : Information

Explanation
The MQSC DISPLAY QUEUE command completed successfully, and details follow this message.

Response
None.

AMQ8410
Parser Error.

Severity
0 : Information

Explanation
The MQSC Parser has an internal error.

Response
None.

AMQ8411
Duplicate Keyword Error.

Severity
0 : Information

Explanation
A command in the MQSC script contains duplicate keywords.

Response
None.

AMQ8412
Numeric Range Error.

Severity
0 : Information

Explanation
The value assigned to an MQSC command keyword is out of the permitted range.

Response
None.

AMQ8413
String Length Error.

Severity
0 : Information

Explanation
A string assigned to an MQSC keyword is either NULL, or longer than the maximum permitted for that keyword.

Response
None.

AMQ8414
Display Channel details.

Severity
0 : Information

Explanation
The MQSC DISPLAY CHL command completed successfully, and details follow this message.

Response
None.

AMQ8415
Ping IBM WebSphere MQ Queue Manager command complete.

Severity
0 : Information

Explanation
The MQSC PING QMGR command completed successfully.

Response
None.

AMQ8416
MQSC timed out waiting for a response from the command server.

Severity
0 : Information

Explanation
MQSC did not receive a response message from the remote command server in the time specified.

Response
None.

AMQ8417
Display Channel Status details.

Severity
0 : Information

Explanation
The MQSC DISPLAY CHANNEL STATUS command completed successfully, and details follow this message.

Response
None.

AMQ8418
<*insert_1*> command responses received.

Severity
0 : Information

Explanation
Running in queued mode, <*insert_1*> command responses were received from the remote command server.

Response
None.

AMQ8419
The Queue is already in the DCE cell.

Severity
0 : Information

Explanation
The Queue is already in the cell, that is, its SCOPE attribute is already CELL.

Response
None.

AMQ8420
Channel Status not found.

Severity
0 : Information

Explanation
No status was found for the specified channel(s).

Response
None.

AMQ8421
A required keyword was not specified.

Severity
0 : Information

Explanation
A keyword required in this command was not specified.

Response
None.

AMQ8422
MQSC found the following response to a previous command on the reply queue :-

Severity
0 : Information

Explanation
MQSC found additional command responses on the reply queue. They will follow this message.

Response
None.

AMQ8423
Cell Directory not available.

Severity
0 : Information

Explanation
The DCE cell directory is not available, so the requested operation has failed.

Response
None.

AMQ8424
Error detected in a name keyword.

Severity
0 : Information

Explanation
A keyword in an MQSC command contained a name string which was not valid. This may be because it contained characters which are not accepted in MQ names. Typical keywords which can produce this error are QLOCAL (and the other q types), CHANNEL, XMITQ, INITQ, MCANAME etc.

Response
None.

AMQ8425
Attribute value error.

Severity
0 : Information

Explanation
A keyword in an MQSC command contained a value that was not valid.

Response
None.

AMQ8426
Valid MQSC commands are:

Severity
0 : Information

Explanation
The text shows valid MQSC commands.

Response
None.

AMQ8427
Valid syntax for the MQSC command:

Severity
0 : Information

Explanation
The text shown is the valid syntax for the MQSC command.

Response
None.

AMQ8428
TYPE Keyword has already been specified.

Severity
0 : Information

Explanation
The TYPE has already been specified after the DISPLAY verb, for example DISPLAY QUEUE(*) type(QLOCAL) type(QALIAS).

Response
Delete the second TYPE keyword and run the command again.

AMQ8429 (IBM i)
Error detected in a exit parameter.

Severity
0 : Information

Explanation
A syntax error occurred an the exit parameter. This may be because it contained characters which are not accepted as exit names. Check the parameters in the MSGEXIT, RCVEXIT, SCYEXIT and SENDEXIT definitions.

Response
None.

AMQ8430
Remote queue manager name is unknown.

Severity
0 : Information

Explanation
The Remote queue manager name is not known to this queue manager. Check that a transmission queue of the same name as the remote queue manager name exists.

Response
Create a transmission queue of the same name as the remote queue manager if one does not exist.

AMQ8431
Transmission queue does not exist

Severity
0 : Information

Explanation
The transmission queue does not exist on this queue manager.

Response
None.

AMQ8432
You are not allowed to set both the REPOS and REPOSNL fields.

Severity
0 : Information

Explanation
An attempt to set both the REPOS and REPOSNL fields has been made. Only one of these fields can have a value other than blank. Both of the fields may be blank.

Response
None.

AMQ8433
You are not allowed to set both the CLUSTER and CLUSNL fields.

Severity
0 : Information

Explanation
An attempt to set both the CLUSTER and CLUSNL fields has been made. Only one of these fields can have a value other than blank. Both of the fields may be blank.

Response
None.

AMQ8434
The repository is unavailable.

Severity
0 : Information

Explanation
The repository is unavailable and the data cannot be accessed. Stop and restart the queue manager.

Response
None.

AMQ8435
All valid MQSC commands were processed.

Severity
0 : Information

Explanation
The MQSC script contains no commands that failed to process.

Response
None.

AMQ8436
One valid MQSC command could not be processed.

Severity
0 : Information

Explanation
The MQSC script contains one command that failed to process.

Response
None.

AMQ8437
No MQSC commands read.

Severity
0 : Information

Explanation
The MQSC script contains no commands.

Response
None.

AMQ8438
One MQSC command read.

Severity
0 : Information

Explanation
The MQSC script contains one command.

Response
None.

AMQ8439
No commands have a syntax error.

Severity
0 : Information

Explanation
The MQSC script contains no commands having a syntax error.

Response
None.

AMQ8440
One command has a syntax error.

Severity
0 : Information

Explanation
The MQSC script contains one command which has a syntax error.

Response
None.

AMQ8441
Display Cluster Queue Manager details.

Severity
0 : Information

Explanation
The MQSC DISPLAY CLUSQMGR command completed successfully, and details follow this message.

Response
None.

AMQ8442
USAGE can not be set to XMITQ with either the CLUSTER or CLUSNL fields set.

Severity
0 : Information

Explanation
An attempt has been made to set USAGE to XMITQ when the CLUSTER or CLUSNL field has a value. Change the value of USAGE, or set the CLUSTER and CLUSNL fields to blank, and try the command again.

Response
None.

AMQ8442 (IBM i)
USAGE can not be set to *TMQ with either the CLUSTER or CLUSNL fields set.

Severity
0 : Information

Explanation
An attempt has been made to set USAGE to *TMQ when the CLUSTER or CLUSNL field has a value. Change the value of USAGE, or set the CLUSTER and CLUSNL fields to blank, and try the command again.

Response
None.

AMQ8443
Only the CLUSTER or CLUSNL field may have a value.

Severity
0 : Information

Explanation
An attempt has been made to set both CLUSTER and CLUSNL fields. One and only one of the fields may have a value, the other field must be blank. Change the value of one of the fields to blank and try the command again.

Response
None.

AMQ8444
The CLUSTER or CLUSNL fields must have a value.

Severity
0 : Information

Explanation
Both the CLUSTER and CLUSNL fields are blank. One and only one of the fields may be blank, the other field must be a value. Change one of the fields from blank to a value and try the command again.

Response
None.

AMQ8445
Program cannot open queue manager object.

Severity
30 : Severe error

Explanation
An attempt to open a queue manager object has failed.

Response
See the previously listed messages in the job log.

AMQ8446
Channel is currently active.

Severity
30 : Severe error

Explanation
The requested operation failed because the channel is currently active.

Response
See the previously listed messages in the job log.

AMQ8447
Requested operation on channel <*insert_3*> not valid for this channel type.

Severity
30 : Severe error

Explanation
The operation requested cannot be performed because channel <*insert_3*> is not of a suitable type. For example, only sender, server and cluster-sender channels can be resolved.

Response
Check that the correct operation was requested. If it was, check that the correct channel name was specified.

AMQ8448
Channel <*insert_3*> is not running.

Severity
30 : Severe error

Explanation
A request to stop channel <*insert_3*> has failed because the channel is not running.

Response
Check that the correct operation was requested. If it was, check that the correct channel name was specified.

AMQ8449
Queue <*insert_3*> inhibited for MQGET.

Severity
30 : Severe error

Explanation
An MQGET failed because the queue <*insert_3*> had been previously inhibited for MQGET.

Response
None.

AMQ8450
Display queue status details.

Severity
0 : Information

Explanation
The MQSC DISPLAY QSTATUS command completed successfully. Details follow this message.

AMQ8451 (IBM i)
STATUS(*STOPPED) not allowed with CONNAME specified.

Severity
0 : Information

Explanation
The STATUS(*STOPPED) parameter is not allowed when specifying CONNAME on the ENDMQMCHL command.

Response
Remove the CONNAME parameter from the command or, specify STATUS(*INACTIVE) to end the channel instance for the specified connection name.

AMQ8452 (IBM i)
STATUS(*STOPPED) not allowed with RQMNAME specified.

Severity
0 : Information

Explanation
The STATUS(*STOPPED) parameter is not allowed when specifying RQMNAME on the ENDMQMCHL command.

Response
Remove the RQMNAME parameter from the command or, specify STATUS(*INACTIVE) to end the channel instance for the specified remote queue manager.

AMQ8453
The path <*insert_3*> is invalid

Severity
20 : Error

Explanation
You typed a path which was not syntactically correct for the operating system you are running IBM WebSphere MQ on.

Response
Determine the correct syntax of a path name for the operating system you are running IBM WebSphere MQ on and use this information to type in a valid path.

AMQ8454
Syntax error found in parameter <*insert_3*>.

Severity
20 : Error

Explanation
The data you entered for <*insert_3*> does not conform to the syntax rules laid down by IBM WebSphere MQ for this parameter.

Response
Carefully check the data entered for this parameter in conjunction with the IBM WebSphere MQ Command Reference to determine the cause of error.

AMQ8455
Password length error

Severity
20 : Error

Explanation
The password string length is rounded up by IBM WebSphere MQ to the nearest eight bytes. This rounding causes the total length of the SSLCRYP string to exceed its maximum.

Response
Decrease the size of the password, or of earlier fields in the SSLCRYP string.

AMQ8456
Conflicting parameters in command.

Severity
20 : Error

Explanation
The command contains parameters that cannot be used together.

Response
Refer to the IBM WebSphere MQ Script (MQSC) Command Reference to determine an allowable combination of parameters for this command.

AMQ8457
IBM WebSphere MQ connection stopped.

Severity
0 : Information

Explanation
The STOP CONN command successfully stopped the connection that was specified.

Response
None.

AMQ8458
IBM WebSphere MQ connection not stopped.

Severity
0 : Information

Explanation
The STOP CONN command could not stop the connection that was specified.

Response
None.

AMQ8459
Not Found.

Severity
0 : Information

Explanation
You specified an identifier that was not found. Please try the command again and supply a valid identifier.

Response
None.

AMQ8460
Syntax error in connection identifier.

Severity
0 : Information

Explanation
You specified an invalid connection identifier. A valid connection identifier contains 16 hex characters, where all of the characters in the connection identifier should lie within the range 0-9, a-z or A-Z.

Response
Correct the connection identifier so that it conforms to the above specification.

AMQ8461
Connection identifier not found.

Severity
0 : Information

Explanation
You specified a connection identifier which is not associated with this queue manager.

Response

Correct the connection identifier so that it describes a connection identifier which is associated with this queue manager. Use the command DISPLAY CONN to identify potential connection identifiers to use with this command.

AMQ8462

The required parameter <*insert_3*> is missing.

Severity

20 : Error

Explanation

The command you entered requires the <*insert_3*> parameter, which has not been specified.

Response

Make sure you specify the missing required parameter.

AMQ8463

At least one of <*insert_3*> must be specified.

Severity

20 : Error

Explanation

At least one of the parameters <*insert_3*> must be specified.

Response

Make sure you specify the required parameters.

AMQ8464

IBM WebSphere MQ subscription <*insert_3*> not found.

Severity

30 : Severe error

Explanation

If the command entered was Change or Display, the subscription <*insert_3*> specified does not exist. If the command entered was Copy, the source subscription does not exist. If the command entered was Create, the system default MQ subscription does not exist.

Response

Correct the subscription name or subscription id specified and then try the command again. If you are creating a new subscription, either specify all parameters explicitly or ensure that the system default subscription, SYSTEM.DEFAULT.SUB, exists.

AMQ8465

The <*insert_3*> attribute cannot be modified for an existing Subscription.

Severity

20 : Error

Explanation

The Subscription could not be altered or replaced.

Response

The Subscription could not be altered or replaced. Check that the command only contains changable attributes.

AMQ8466

The remote queue <*insert_3*> could not be opened.

Severity

30 : Severe error

Explanation
The remote queue could not be opened..

Response
Check that the remote Queue is correctly defined on the remote Queue Manager.

AMQ8467
There was a syntax error in the hex string representing the bytes value of a keyword.

Severity
0 : Information

Explanation
The hex string that was entered was found to contain a syntax error. This error may occur for one of the following reasons:

- The string was too long.
- The string contained invalid hex characters.

Valid characters are 0-9, A-F and a-f. Hex strings with an odd number of characters will be prefixed with a zero, for example, DESTCORL(A) will be interpreted as DESTCORL(0A)

Response
None.

AMQ8468
DEST field must not be set when using DESTCLAS(MANAGED)

Severity
30 : Severe error

Explanation
An attempt to set both DESTCLAS(MANAGED) and DEST has been made. When using DESTCLAS(MANAGED) do not specify a destination. If a destination is required then DESTCLAS(PROVIDED) should be used.

Response
None.

AMQ8469
IBM WebSphere MQ subscription <*insert_3*> in use.

Severity
30 : Severe error

Explanation
The subscription <*insert_3*> specified is currently in use by another application.

Response
Ensure that no applications are using the specified subscription, then try the command again.

AMQ8470
The object <*insert_3*> is not a valid subscription destination.

Severity
30 : Severe error

Explanation
The object <*insert_3*> is not of a permitted type for a subscription destination.

Response
If using a QALIAS as a subscription destination object, ensure that its TARGTYPE attribute has the value of QUEUE.

AMQ8471
IBM WebSphere MQ topic string error

Severity
30 : Severe error

Explanation
The topic string (TOPICSTR) supplied was not valid

Response
Correct the topic string definition and try the command again.

AMQ8472
IBM WebSphere MQ topic string not found

Severity
30 : Severe error

Explanation
The topic string supplied does not exist in the topic tree

Response
Correct the topic string used and try the command again

AMQ8473
A IBM WebSphere MQ topic using the supplied topic string already exists

Severity
30 : Severe error

Explanation
The topic string supplied has been specified on a previously created topic object. At most, one topic object per topic string is permitted.

Response
If the topic string specified is incorrect, modify the topic string and retry the operation. Alternatively, if the previously created topic object is not required, delete that topic object first, then retry the operation.

AMQ8474
The required parameter SUB is invalid.

Severity
20 : Error

Explanation
The command you entered requires a valid SUB parameter.

Response
Make sure the required parameter is correct.

AMQ8475
Subscription already exists.

Severity
20 : Error

Explanation
The Subscription <*insert_3*> could not be created because it already exists.

Response
Check that the name is correct and try the command again specifying REPLACE, or delete the Subscription. Then try the command again.

AMQ8476
The required parameter <*insert_3*> is missing.

Severity
20 : Error

Explanation
The command you entered requires the <*insert_3*> parameter, which has not been specified.

Response
Make sure you specify the missing required parameter.

AMQ8477
The specified options are invalid.

Severity
40 : Stop Error

Explanation
The combination of options supplied for the command are invalid.

Response
Check the specified options and ensure they are correct.

AMQ8478
Standby queue manager.

Severity
40 : Stop Error

Explanation
The queue manager is a standby queue manager. You must use the primary instance of a queue manager to administer it.

Response
Re-issue the command on the primary instance of the queue manager.

AMQ8480
Subscription <*insert_3*> could not be created. The reason code from the MQSUB function call was <*insert_1*>.

Severity
20: Error

Explanation
During the attempt to create subscription name '<*insert_3*>', an error was detected. The reason for failure is <*insert_1*>. This reason code is returned from the MQSUB function call.

Response
Check the reason code in the IBM WebSphere MQ Messages manual, correct the underlying problem and then try the command again.

AMQ8482
Cluster topics inhibited due to PSCLUS(DISABLED).

Severity
20: Error

Explanation
The queue manager attribute PSCLUS has been set to DISABLED so clustered topics cannot be defined and existing topics can not be altered to set the CLUSTER attribute. Topic <*insert_3*> has not been created or altered on this system.

Response
If you need to enable publish/subscribe clustering, modify the PSCLUS attribute to ENABLED on all queue managers participating in the cluster.

AMQ8483
Unable to modify PSCLUS because cluster topic(s) exist.

Severity
20: Error

Explanation
Queue manager attribute PSCLUS has been set to DISABLED to indicate that Publish/Subscribe activity is not expected between queue managers in this cluster. However, a cluster topic already exists, so the setting cannot be modified. The PSCLUS attribute remains unchanged.

Response
If you need to disable publish/subscribe activity within this cluster, first DELETE all cluster topic objects, then remodify the PSCLUS attribute.

AMQ8491
Timeout waiting for a reply from the Telemetry service.

Severity
0 : Information

Explanation
Timeout waiting for a reply from the Telemetry Service '`SYSTEM.MQXR.SERVICE`'.

Response
Reduce the number of responses expected from the Telemetry Service by using a **`where`** clause.

AMQ8492
The number of responses has been limited to <*insert_1*>

Severity
0 : Information

Explanation
The number of responses has been limited to the **`MAXDEPTH`** of the ReplyToQueue '`SYSTEM.MQSC.REPLY.QUEUE`'.

Response
Reduce the number of responses expected from the Telemetry Service by using a **`where`** clause, or increase the **`MAXDEPTH`** of the ReplyToQueue '`SYSTEM.MQSC.REPLY.QUEUE`'.

AMQ8498
Starting MQSC for queue manager <*insert_3*>.

Severity
0 : Information

Explanation
The MQSC script contains <*insert_1*> commands.

Response
None.

AMQ8499
Usage: runmqsc [-e] [-v] [-w WaitTime [-x]] [QMgrName]

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8499 (Tandem)
Usage: runmqsc [-e] [-v] [-w WaitTime] [-x] [-i In] [-o Out] QMgrName

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8500
IBM WebSphere MQ Display MQ Files

Severity
0 : Information

AMQ8501
Common services initialization failed with return code <*insert_1*>.

Severity
20 : Error

Explanation
A request by the command server to initialize common services failed with return code <*insert_1*>.

Response
None.

AMQ8502
Connect shared memory failed with return code <*insert_1*>.

Severity
20 : Error

Explanation
A request by the command server to connect shared memory failed with return code <*insert_1*>.

Response
None.

AMQ8503
Post event semaphore failed with return code <*insert_1*>.

Severity
20 : Error

Explanation
A request by the command server to post an event semaphore failed with return code <*insert_1*>.

Response
None.

AMQ8504
Command server MQINQ failed with reason code <*insert_1*>.

Severity
20 : Error

Explanation
An MQINQ request by the command server, for the IBM WebSphere MQ queue <*insert_3*>, failed with reason code <*insert_1*>.

Response
None.

AMQ8505
Reallocate memory failed with return code <*insert_1*>.

Severity
20 : Error

Explanation
A request by the command server to reallocate memory failed with return code <*insert_1*>.

Response
None.

AMQ8506
Command server MQGET failed with reason code <*insert_1*>.

Severity
20 : Error

Explanation
An MQGET request by the command server, for the IBM WebSphere MQ queue <*insert_3*>, failed with reason code <*insert_1*>.

Response
None.

AMQ8507
Command server MQPUT1 request for an undelivered message failed with reason code <*insert_1*>.

Severity
20 : Error

Explanation
An attempt by the command server to put a message to the dead-letter queue, using MQPUT1, failed with reason code <*insert_1*>. The MQDLH reason code was <*insert_2*>.

Response
None.

AMQ8508
Queue Manager Delete Object List failed with return code <*insert_1*>.

Severity
20 : Error

Explanation
A request by the command server to delete a queue manager object list failed with return code <*insert_1*>.

Response
None.

AMQ8509
Command server MQCLOSE reply-to queue failed with reason code <*insert_1*>.

Severity
20 : Error

Explanation
An MQCLOSE request by the command server for the reply-to queue failed with reason code <*insert_1*>.

Response
None.

AMQ8510
Command server queue is open, try again later.

Severity
30 : Severe error

AMQ8511
Usage: strmqcsv [QMgrName]

Severity
0 : Information

AMQ8512
Usage: endmqcsv [-c | -i] QMgrName

Severity
0 : Information

AMQ8513
Usage: dspmqcsv [QMgrName]

Severity
0 : Information

AMQ8514
No response received after <*insert_1*> seconds.

Severity
20 : Error

Explanation
The command server has not reported the status of running, to the start request, before the timeout of <*insert_1*> seconds was reached.

Response
None.

AMQ8515 (Tandem)
MQSeries Alter MQ Files

Severity
0 : Information

Explanation
Title for the altmqfls command.

Response
None.

AMQ8516 (Tandem)
MQSeries Clean Queue Manager

Severity
0 : Information

Explanation
Title for the cleanqm command.

Response
None.

AMQ8517 (Tandem)
The messages files are partitioned and cannot be moved.

Severity
0 : Information

Explanation
Partition Error from the altmqfls command.

Response
None.

AMQ8518
LOGGEREV is only valid when using a linear logging queue manager.

Severity
20 : Error

Explanation
The LOGGEREV attribute may only be set to ENABLED when the queue manager was created as a linear logging queue manager. For more information about logging, see Making sure that messages are not lost (logging).

Response
The system admistrator should only attempt to change the LOGGEREV queue manager attribute when the queue manager being administered was created as a linear logging queue manager.

AMQ8519
The topic object <*insert_3*> does not permit durable subscription.

Severity
30 : Severe error

Explanation
The topic object <*insert_3*> has been defined to disallow durable subscription.

Response
Ensure that the topic object to which you are creating a subscription allows durable subscription.

AMQ8520
The queue name supplied is not valid for DEFXMITQ.

Severity
20 : Error

Explanation
The specified queue is not allowed to be used as the default transmission queue because it is reserved for use exclusively by clustering.

Response
Change the value of DEFXMITQ, and try the command again.

AMQ8549
Total string length exceeds the maximum value of 999 characters.

Severity
0 : Information

Explanation
The total length of a channel exit string is 999 characters. The string list assigned to an MQSC keyword is longer than the maximum value of 999 characters permitted for that keyword.

Response
None.

AMQ8550
Display namelist details.

Severity
0 : Information

Explanation
The MQSC DISPLAY NAMELIST command completed successfully, and details follow this message.

Response
None.

AMQ8551
IBM WebSphere MQ namelist changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ namelist <*insert_5*> changed.

Response
None.

AMQ8552
IBM WebSphere MQ namelist created.

Severity
0 : Information

Explanation
IBM WebSphere MQ namelist <*insert_5*> created.

Response
None.

AMQ8553
IBM WebSphere MQ namelist deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ namelist <*insert_5*> deleted.

Response
None.

AMQ8554
String List String Count Error.

Severity
0 : Information

Explanation
The number of strings within the stringlist is greater than the maximum number allowed for the keyword. Reduce the number of strings within the list and try the command again.

Response
None.

AMQ8555
String List String Length Error.

Severity
0 : Information

Explanation
A string in a string list assigned to a keyword is longer than the maximum permitted for that keyword.

Response
None.

AMQ8556
RESUME QUEUE MANAGER accepted.

Severity
0 : Information

Explanation
The RESUME QUEUE MANAGER command has been accepted for processing. The command will be sent to the repository which will process the command and notify all other repositories that this queue manager is now back in the cluster.

Response
None.

AMQ8557
SUSPEND QUEUE MANAGER accepted.

Severity
0 : Information

Explanation
The SUSPEND QUEUE MANAGER command has been accepted for processing. The command will be sent to the repository which will process the command and notify all other repositories that this queue manager is leaving the cluster.

Response
None.

AMQ8558
REFRESH CLUSTER accepted.

Severity
0 : Information

Explanation
The REFRESH CLUSTER command has been accepted for processing. The command will be sent to the Repository which will process the command and notify all other repositories that the Cluster needs refreshing.

Response
None.

AMQ8559
RESET CLUSTER accepted.

Severity
0 : Information

Explanation
The RESET CLUSTER command has been accepted for processing. The command will be sent to the Repository which will process the command and notify all other repositories that the Cluster needs resetting.

Response
None.

AMQ8560
IBM WebSphere MQ security cache refreshed.

Severity
0 : Information

Explanation
The object authority manager security cache has been refreshed.

Response
None.

AMQ8561 (Tandem)
IBM WebSphere MQ for HP Integrity NonStop Server does not support this option.

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8561 (Windows)
Domain controller unavailable.

Severity
10 : Warning

Explanation
IBM WebSphere MQ was unable to contact the domain controller to obtain information for user <*insert_3*>.

Response
Ensure that a domain controller for the domain on which user <*insert_3*> is defined is available. Alternatively, if you are using a computer which is not currently connected to the network and have logged on using a domain user ID, you may wish to log on using a local user ID instead.

AMQ8562
The Java application failed to connect to the Queue Manager because the version of the native JNI library <*insert_3*> is inconsistant with the version of the IBM WebSphere MQ Queue Manager <*insert_4*>.

Severity
10 : Warning

Explanation
The native JNI library <*insert_3*> is out-of-date compared to the IBM WebSphere MQ Queue Manager <*insert_4*>

Response
Ensure that the Java library path points to the current version of the JNI library

AMQ8562 (Tandem)
Command line does not exist

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8563
IBM WebSphere MQ authentication information object created.

Severity
0 : Information

Explanation
IBM WebSphere MQ authentication information object <*insert_3*> created.

Response
None.

AMQ8564
IBM WebSphere MQ authentication information object deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ authentication information object <*insert_3*> deleted.

Response
None.

AMQ8565
Queue Status not found.

Severity
0 : Information

Explanation
Queue Status for the specified queue could not be found.

Response
None.

AMQ8566
Display authentication information details.

Severity
0 : Information

Explanation
The MQSC DISPLAY AUTHINFO command completed successfully. Details follow this message.

Response
None.

AMQ8567
IBM WebSphere MQ authentication information changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ authentication information <*insert_3*> changed.

Response
None.

AMQ8568
The native JNI library <*insert_3*> was not found.

Severity
10 : Warning

Explanation
The native JNI library <*insert_3*> could not be loaded because the library was not found.

Response
Ensure that the java library path points to the location of the JNI library.

AMQ8568 (IBM i)
No authinfo objects to display.

Severity
0 : Information

Explanation
There are no matching authinfo objects defined on this system.

Response
Using the DEFINE AUTHINFO command to create an authinfo object.

AMQ8569
Error in filter specification

Severity
0 : Information

Explanation
You specified an invalid filter. Check the WHERE statement and make sure that the operator is valid for the type of parameter, that the parameter can be filtered on, and that the value that you specified for the filter is valid for the type of attribute you are filtering on.

Response
None.

AMQ8570
Attribute value error in <*insert_3*>.

Severity
0 : Information

Explanation
The keyword <*insert_3*> contained a value that was not valid for this configuration. Please check the MQSC Command Reference to determine valid values for <*insert_3*>.

Response
None.

AMQ8571
<*insert_1*> authority not revoked from the <*insert_2*> group for reason "1111".

Severity
10 : Warning

Explanation
As part of queue manager migration an attempt was made to revoke <*insert_1*> authority from the <*insert_2*> group for the <*insert_3*> object. That attempt failed for reason "1111".

Response
An administrator must determine the cause of failure and then use the **setmqaut** command to manually revoke <*insert_1*> authority from the <*insert_2*> group for the <*insert_3*> object.

AMQ8572
Securing IBM WebSphere MQ objects against local groups may yield undesirable results.

Severity
10 : Warning

Explanation
A request was made to secure a IBM WebSphere MQ object against a local group in a Multi Instance queue manager environment. Access to these objects may be refused upon switch-over.

Response

An administrator should determine whether the request was intentional and use the setmqaut command to secure the IBM WebSphere MQ object against a corresponding domain group.

AMQ8574

Refreshing settings for primary installation "<*insert_1*>" (<*insert_2*>)

Severity

10 : Warning

Explanation

A request was issued to set installation "<*insert_1*>" as the primary installation however this installation is already set as the Primary Installation. The command continues and refreshes the settings which identify this installation as the primary installation.

Response

None.

AMQ8575

Unable to access installation task file "<*insert_1*>".

Severity

20 : Error

Explanation

An attempt was made to access the IBM WebSphere MQ installation task file "<*insert_1*>" however the command issued was unable to access the file.

Response

Further messages might have been issued giving more details about the failure to access the file. Check that the file exists, and the access permissions are correct. Correct any errors and re-issue the command.

AMQ8576

"<*insert_1*>" (<*insert_2*>) set as the primary installation. You must restart the operating system to complete the update.

Severity

0 : Information

Explanation

All tasks required to set installation "<*insert_1*>" as the primary installation have been completed. If the installation was not already set as the primary installation then the installation configuration has also been updated to identify installation "<*insert_1*>" as the primary installation.

In order to ensure that the updates are visible machine-wide, a restart of the operating system is required.

Response

None.

AMQ8577

Failed to set "<*insert_1*>" (<*insert_2*>) as the primary installation.

Severity

20 : Error

Explanation

The command attempted to set installation "<*insert_1*>" as the primary installation but one or more of the tasks required to set the installation as the primary installation failed to complete successfully. Any updates made by the command have been undone.

Response
Further messages have been issued giving more details about the failure. Correct any identified errors and re-issue the command.

AMQ8578
Failed to refresh configuration for primary installation "<*insert_1*>" (<*insert_2*>).

Severity
20 : Error

Explanation
The command attempted to refresh the tasks required to set installation "<*insert_1*>" as the primary installation but one or more of the tasks failed to complete successfully. Installation "<*insert_1*>" is still set as the primary installation.

Response
Further messages have been issued giving more details about the failure. Correct any identified errors and re-issue the command.

AMQ8579
Primary installation cannot be changed from "<*insert_2*>" to "<*insert_1*>".

Severity
20 : Error

Explanation
The command attempted to set installation "<*insert_1*>" as the primary installation but the operation could not be performed because installation "<*insert_2*>" is already set as the primary installation.

Response
In order to set installation "<*insert_1*>" as the primary installation you must first unset installation "<*insert_2*>" as the primary installation using the command "**setmqinst -x -n** <*insert_2*>". You can then re-issue the command to set installation "<*insert_1*>" as the primary installation.

AMQ8580
Failed to unset "<*insert_1*>" (<*insert_2*>) as the primary installation.

Severity
20 : Error

Explanation
The command attempted to unset installation "<*insert_1*>" as the primary installation but one or more of the tasks required to unset the installation as the primary installation failed to complete successfully. The installation remains set as the primary installation.

Response
Further messages have been issued giving more details about the failure. Correct any identified errors and re-issue the command.

AMQ8581
"<*insert_1*>" (<*insert_2*>) is not currently set as the primary installation.

Severity
20 : Error

Explanation
The command attempted to unset installation "<*insert_1*>" as the primary installation but installation "<*insert_1*>" is not currently set as the primary installation.

Response
Verify the name of the installation supplied is correct and re-issue the command if necessary.

AMQ8582
"<*insert_1*>" (<*insert_2*>) has been unset as the primary installation.

Severity
0 : Information

Explanation
All tasks required to unset installation "<*insert_1*>" as the primary installation have been completed.

Response
None

AMQ8583
Installation details for <*insert_3*> location <*insert_4*> missing or corrupt.

Severity
20 : Error

Explanation
The command attempted to access the installation details for installation <*insert_3*> location <*insert_4*> but the installation details were not found or are corrupt.

Response
Use the dspmqinst command to verfiy the contents of the installation configuration file. If the entry is missing or corrupt use the crtmqinst command with the -r parameter to rebuild the configuration information for the installation.

AMQ8584
Insufficient permission to update installation configuration.

Severity
20 : Error

Explanation
An attempt was made to update the IBM WebSphere MQ installation configuration for Installation <*insert_3*> location <*insert_4*> but the request was rejected as the current user does not have sufficient authority to make the update.

Response
Issue the command from a user with sufficient authority to update the installation configuration.

AMQ8585
Invalid value specified for <*insert_3*> parameter.

Severity
20 : Error

Explanation
The value supplied for the <*insert_3*> parameter is invalid.

Response
Verify that the value supplied is

- correctly specified
- contains only valid characters
- does not exceed the maxmium length for the parameter

AMQ8586
Usage: `setmqinst (-n InstName | -p InstPath) (-i | -x | -d Text)`

```
-d   Descriptive text.
-i   Set this installation as the primary installation.
-n   Installation name.
```

```
-p   Installation path.
-x   Unset this installation as the primary installation.
```

Severity
0 : Information

Explanation
This message shows correct usage.

Response
None.

AMQ8587
Note there are a number (1111) of other installations, use the "**-i**" parameter to display them.

Severity
0 : Information

Explanation

Response
None.

AMQ8588
No parameter was detected. The environment has been set for the installation from which the **setmqenv** command was issued.

Severity
10 : Warning

Explanation
The environment has been set for the installation that **setmqenv** originates from because **setmqenv** detected no parameters. If you specified parameters but these parameters have been ignored, it might be because the shell script you are using cannot pass parameters to a sourced script.

Response
If you intended to set up the environment for another installation but did not specify any parameters, issue the command again specifying the correct parameters. If you specified parameters for **setmqenv** but they have been ignored, use the **setmqenv** command from the installation you want to set up the environment for. Use the **dspmqinst** command to determine the path for other installations and use the **dspmq** command to determine the installation associated with a specific queue manager.

AMQ8589
Installation "<*insert_1*>" (<*insert_2*>) is implicitly primary.

Severity
10: Warning

Explanation
The command attempted to modify the primary installation "<*insert_1*>", however this installation is implicitly primary and can only be made non-primary by uninstalling this installation.

Response
Verify that the installation "<*insert_1*>" is required, if so no other installation can be made primary.

AMQ8590
Installation "<*insert_1*>" (<*insert_2*>) is not installed.

Severity
20 : Error

Explanation
A command was issued specifying an installation which is not currently installed. The installation must be installed for this command to run.

Response
None.

AMQ8592
Queue manager "<*insert_1*>" is now associated with installation "<*insert_2*>"

Severity
0: Information

Explanation
A command was issued that has associated queue manager "<*insert_1*>" with installation "<*insert_2*>". The queue manager is executed by this installation when it is next started.

Response
None

AMQ8593
Installation state for installation "<*insert_1*>" ("<*insert_2*>") detected as invalid.

Severity
20 : Error

Explanation
An attempt was made to modify the state of installation "<*insert_1*>" ("<*insert_2*>"), however an error was detected related to the current state of this installation which prevented the change from occurring.

Response
Investigate recent changes to the system that might have invalidated installation "<*insert_1*>". It might be necessary to contact your IBM support center, in which case a trace of the failing command might be required.

AMQ8595
The **setmqenv** command was not preceded by the **source** command.

Severity
20 : Error

Explanation
The command script containing **setmqenv** modifies the environment of the shell in which it is running. Because you did not precede **setmqenv** with the source command, it runs in a new shell and it modifies the environment in the new shell. When the **setmqenv** command ends, the new shell ends and control returns to the old shell. The old shell does not inherit changes to the environment from the new shell. The result is that the environment of the old shell, containing the **setmqenv** command does not change.

Response
Precede **setmqenv** with the **source** command. The combination of a dot followed by a space is a synonym for the source command; for example:

```
. setmqenv -s
```

AMQ8597
This process can only use installation "<*insert_4*>".

Severity
10 : Error

Explanation
An MQ_long shared library "<*insert_3*>" was detected in this process before the first connection to a queue manager was made.

Linking applications to this shared library is deprecated. Applications that do so should be re-linked because it inhibits the use of multiple installations from within the application.

As a temporary work-around, this process is allowed to connect to queue managers associated with installation "<*insert_4*>". Attempting to connect to a queue manager associated with an installation other than "<*insert_4*>" will either fail with reason code MQRC_INSTALLATION_MISMATCH or MQRC_FASTPATH_NOT_AVAILABLE.

To obtain full multiple installation functionality, you must re-link this application, omitting -lmqmcs and -lmqmzse from the link step.

Response

Re-link your application, omitting the -lmqmcs and -lmqmzse options from the command line. When the application is re-linked without libmqmcs or libmqmzse, these restrictions are lifted and the application supports connecting to queue managers from installations other than "<*insert_4*>".

This message can be suppressed by setting the AMQ_NO_MQMCS_MSG environment variable to any value.

AMQ8601

IBM WebSphere MQ trigger monitor started.

Severity

0 : Information

Explanation

The IBM WebSphere MQ trigger monitor has been started.

Response

None.

AMQ8601 (IBM i)

IBM WebSphere MQ trigger monitor started.

Severity

0 : Information

Explanation

The trigger monitor has been started with initiation queue <*insert_3*>.

Response

None.

AMQ8602

IBM WebSphere MQ trigger monitor ended with exit code <insert_1>. If this value is anything other than zero, it indicates an error condition.

Severity

0 : Information

Explanation

The IBM WebSphere MQ trigger monitor has ended with exit code <insert_1>.

Response

Look for earlier error messages from the trigger monitor.

AMQ8603

Usage: runmqtrm [-m QMgrName] [-q InitQ]

Severity

0 : Information

Explanation

None.

Response
None.

AMQ8604
Use of IBM WebSphere MQ trigger monitor not authorized.

Severity
0 : Information

Explanation
The trigger monitor cannot be run due to lack of authority to the requested queue manager or initiation queue.

Response
Obtain the necessary authority from your security officer or IBM WebSphere MQ administrator. Then try the command again.

AMQ8605
Queue manager not available to the IBM WebSphere MQ trigger monitor

Severity
0 : Information

Explanation
The queue manager specified for the trigger monitor does not exist, or is not active.

Response
Check that you named the correct queue manager. Ask your systems administrator to start it, if it is not active. Then try the command again.

AMQ8606
Insufficient storage available for the IBM WebSphere MQ trigger monitor.

Severity
0 : Information

Explanation
There was insufficient storage available for the IBM WebSphere MQ trigger monitor to run.

Response
Free some storage and then try the command again.

AMQ8607
IBM WebSphere MQ trigger monitor connection failed.

Severity
0 : Information

Explanation
The trigger monitor's connection to the requested queue manager failed because of MQI reason code <*insert_1*> from MQCONN.

Response
Consult your systems administrator about the state of the queue manager.

AMQ8608
IBM WebSphere MQ trigger monitor connection broken.

Severity
0 : Information

Explanation
The connection to the queue manager failed while the trigger monitor was running. This may be caused by an endmqm command being issued by another user, or by a queue manager error.

Response
Consult your systems administrator about the state of the queue manager.

AMQ8609
Initiation queue missing or wrong type

Severity
0 : Information

Explanation
The named initiation queue could not be found; or the queue type is not correct for an initiation queue.

Response
Check that the named queue exists, and is a local queue, or that the named queue is an alias for a local queue which exists.

AMQ8610
Initiation queue in use

Severity
0 : Information

Explanation
The IBM WebSphere MQ trigger monitor could not open the initiation queue because the queue is open for exclusive use by another application.

Response
Wait until the queue is no longer in use, and try the command again.

AMQ8611
Initiation queue could not be opened.

Severity
0 : Information

Explanation
The IBM WebSphere MQ trigger monitor could not open the initiation queue; reason code <*insert_1*> was returned from MQOPEN.

Response
Consult your systems administrator.

AMQ8612
Waiting for a trigger message

Severity
0 : Information

Explanation
The IBM WebSphere MQ trigger monitor is waiting for a message to arrive on the initiation queue.

Response
None.

AMQ8613
Initiation queue changed or deleted

Severity
0 : Information

Explanation
The IBM WebSphere MQ trigger monitor is unable to continue because the initiation queue has been deleted or changed since it was opened.

Response
Retry the command.

AMQ8614
Initiation queue not enabled for input.

Severity
0 : Information

Explanation
The IBM WebSphere MQ trigger monitor cannot read from the initiation queue because input is not enabled.

Response
Ask your systems administrator to enable the queue for input.

AMQ8615
IBM WebSphere MQ trigger monitor failed to get message.

Severity
0 : Information

Explanation
The IBM WebSphere MQ trigger monitor failed because of MQI reason code <*insert_1*> from MQGET.

Response
Consult your systems administrator.

AMQ8616
End of application trigger.

Severity
0 : Information

Explanation
The action to trigger an application has been completed.

Response
None.

AMQ8617
Not a valid trigger message.

Severity
0 : Information

Explanation
The IBM WebSphere MQ trigger monitor received a message that is not recognized as a valid trigger message. If the queue manager has a dead letter queue, the trigger monitor attempts to put the message onto that queue. If that operation succeeds, the trigger monitor continues. Otherwise, the trigger monitor checks whether the Report options in the message descriptor allow the message to be discarded. If so, the message is discarded and the trigger monitor continues. If not, the operation is backed out and the trigger monitor ends.

Response
Investigate the reason why the trigger message was invalid. Check that you have started the trigger monitor to consume from the correct queue. The trigger monitor must be given the name of an initiation queue, not an application queue. If you have started it to consume from an application queue, this should be corrected.

AMQ8618
Error <*insert_1*> starting triggered application (errno <*insert_2*>).

Severity
0 : Information

Explanation
An error was detected when trying to start the application identified in a trigger message. The system() call returned <*insert_1*>. This can cause the value of errno to be set. In this case the value was <*insert_2*>.

Response
Check that the application the trigger monitor was trying to start is available. Refer to documentation for the system() call as to why the triggered application failed to start.

AMQ8619
Application type <*insert_1*> not supported.

Severity
0 : Information

Explanation
A trigger message was received which specifies application type <*insert_1*>; the trigger monitor does not support this type.

Response
Use an alternative trigger monitor for this initiation queue.

AMQ8620
Trigger message with warning <*insert_1*>

Severity
0 : Information

Explanation
The trigger monitor received a message with a warning. For example, it may have been truncated or it could not be converted to the trigger monitor's data representation. The reason code for the warning is <*insert_1*>.

Response
None.

AMQ8621
Usage: runmqtmc [-m QMgrName] [-q InitQ]

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8622
Usage: CICS-Transaction-Name [MQTMC2 structure]

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8623
IBM WebSphere MQ listener changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ listener <*insert_3*> changed.

Response
None.

AMQ8624
IBM WebSphere MQ service changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ service <*insert_3*> changed.

Response
None.

AMQ8625
IBM WebSphere MQ service created.

Severity
0 : Information

Explanation
IBM WebSphere MQ service <*insert_3*> created.

Response
None.

AMQ8626
IBM WebSphere MQ listener created.

Severity
0 : Information

Explanation
IBM WebSphere MQ listener <*insert_3*> created.

Response
None.

AMQ8627
IBM WebSphere MQ service object deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ service object <*insert_3*> deleted.

Response
None.

AMQ8628
IBM WebSphere MQ listener object deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ listener object <*insert_3*> deleted.

Response
None.

AMQ8629
Display service information details.

Severity
0 : Information

Explanation
The MQSC DISPLAY SERVICE command completed successfully. Details follow this message.

Response
None.

AMQ8630
Display listener information details.

Severity
0 : Information

Explanation
The MQSC DISPLAY LISTENER command completed successfully. Details follow this message.

Response
None.

AMQ8631
Display listener status details.

Severity
0 : Information

Explanation
The MQSC DISPLAY LSSTATUS command completed successfully. Details follow this message.

AMQ8632
Display service status details.

Severity
0 : Information

Explanation
The MQSC DISPLAY SVSTATUS command completed successfully. Details follow this message.

AMQ8633
Display topic details.

Severity
0 : Information

Explanation
The MQSC DISPLAY TOPIC command completed successfully. Details follow this message.

AMQ8634 (Tandem)
Message Overflow file could not be created for queue <*insert_1*>

Severity
0 : Information

Explanation
When attempting to create a file to hold a large message (a message larger than the message overflow threshold for the queue) the Queue Manager was unable to identify a unique filename for the file. This is probably caused by too many existing large messages for the queue, or for the queue manager as a whole if the default location for large message storage is being used.

Response
Use altmqfls to change the subvolume for large message storage for this Queue.

AMQ8635 (Tandem)
A Queue Server has ended normally.

Severity
0 : Information

Explanation
A Queue Server in CPU <*insert_1*> has ended normally. The process was named <*insert_3*>.

Response
None.

AMQ8636 (Tandem)
A Queue Server has ended with errors.

Severity
0 : Information

Explanation
A Queue Server in CPU <*insert_1*> has ended with errors. The process was named <*insert_3*>. The error return code reported by the Queue Server is <*insert_2*>. The Queue Server should be restarted automatically by the Queue Manager.

Response
Verify that the Queue Server has restarted correctly. Examine the Queue Manager FD subvolume for FFST files that may have been generated by the Queue Server. Use the process name to locate the relevant FFSTs. Attempt to reconstruct the chain of events or symptoms that lead to the failure and save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8637 (Tandem)
A Queue Server has detected a CPU failure.

Severity
0 : Information

Explanation
The Queue Server process <*insert_3*> has detected that CPU <*insert_1*> failed. If there were components of the Queue Manager that were running in this CPU, they will now no longer be available, and application connections and channels may be dropped. The Queue Manager should continue to be available to new connections and channels. Any Status Server and Queue Server processes that were running in that CPU will be replaced in other available CPUs.

Response
None normally necessary. Applications could experience the reason code MQRC_CONNECTION_BROKEN (2009) from MQI operations in progress that used agent processes running in the failed CPUs, but they should be able to immediatelt re-connect successfully.

AMQ8638 (Tandem)
A Queue Server completed takeover processing.

Severity
0 : Information

Explanation
The Queue Server process <*insert_3*> has completed processing that was associated with a prior

takeover from a failed primary Queue Server process, or the failure of the CPU that it was running in. Normal processing resumes after this point, and the Queue Server is again in a state where it is resilient to any single point of failure.

Response
None normally necessary. This message is logged to provide positive confirmation that the takeover is complete.

AMQ8639 (Tandem)
A Queue Server processed expired messages.

Severity
0 : Information

Explanation
The Queue Server process <*insert_3*> detected and processed <*insert_1*> messages that have expired.

Response
None normally necessary. This message is logged to provide information about the number of messages that expire for each Queue Server. If performance degradation is experienced for a particular Queue Server, verify that there are not an excessively large number of expired messages having to be processed by that Queue Server process.

AMQ8640 (Tandem)
Signal delivery timeout expired for an MQGET.

Severity
0 : Information

Explanation
The Queue Server process <*insert_3*> failed to open and send a signal to the application process <*insert_4*> within the timeout allowed for signal delivery. The MQGET with the MQGMO_SET_SIGNAL option issued by the application has been canceled by the Queue Server, but no notification can be delivered to the application.

Response
Manual intervention with the application may be necessary to ensure that it resumes normal processing. No further notification will be delivered to the application relating to the MQGET call that established the signal. The application can re-open the queue and re-issue the MQGET call to recover from this situation.

AMQ8641 (Tandem)
Signal delivery open error for an MQGET.

Severity
0 : Information

Explanation
The Queue Server process <*insert_3*> failed to open the application process <*insert_4*> in order to deliver a signal IPC. The file system error number was <*insert_1*>. The MQGET with the MQGMO_SET_SIGNAL option issued by the application has been canceled by the Queue Server, but no notification can be delivered to the application.

Response
Manual intervention with the application may be necessary to ensure that it resumes normal processing. No further notification will be delivered to the application relating to the MQGET call that established the signal. The application can re-open the queue and re-issue the MQGET call to recover from this situation.

AMQ8642 (Tandem)
Signal delivery error for an MQGET.

Severity
0 : Information

Explanation
The Queue Server process <*insert_3*> failed to deliver a signal IPC to the application process <*insert_4*>. The file system error number was <*insert_1*>. The MQGET with the MQGMO_SET_SIGNAL option issued by the application has been canceled by the Queue Server, but no notification can be delivered to the application.

Response
Manual intervention with the application may be necessary to ensure that it resumes normal processing. No further notification will be delivered to the application relating to the MQGET call that established the signal. The application can re-open the queue and re-issue the MQGET call to recover from this situation.

AMQ8643 (Tandem)
Signal delivery canceled for an MQGET.

Severity
0 : Information

Explanation
The Queue Server process <*insert_3*> was required to terminate an MQGET with the MQGMO_SET_SIGNAL option before the specified Waitinterval expired but failed to open the application process <*insert_4*> in order to deliver a signal IPC. The MQGET with the MQGMO_SET_SIGNAL option issued by the application has been canceled by the Queue Server, but no notification can be delivered to the application.

Response
Manual intervention with the application may be necessary to ensure that it resumes normal processing. No further notification will be delivered to the application relating to the MQGET call that established the signal. The application can re-open the queue and re-issue the MQGET call to recover from this situation.

AMQ8644 (Tandem)
Queue Server memory threshold exceeded.

Severity
0 : Information

Explanation
The Queue Server process <*insert_3*> reached the threshold memory usage (<*insert_1*> bytes) at which unused queues are eligible for unloading to disk.

Response
Verify that the Queue Server is not overloaded with queues, or that messages are not building up unexpectedly on queues supported by the Queue Server.

AMQ8645 (Tandem)
Memory usage for Queue Server now below threshold.

Severity
0 : Information

Explanation
The memory usage of Queue Server process <*insert_3*> has now reduced to below the threshold (<*insert_1*> bytes) at which unused queues are unloaded to disk.

Response
None.

AMQ8646 (Tandem)
NonStop TM/MP reports transactions disabled

Severity
0 : Information

Explanation
The Queue Server <*insert_3*> has detected that the Compaq NonStop TM/MP has disabled transactions on the NSK system. The Queue Servers in the Queue Manager will no longer accept MQPUT or non-browse MQGET operations on Persistent messages, or any sync point operation. Attempts to perform operations on persistent messages will be rejected with teh reason code MQRC_SYNCPOINT_NOT_AVAILABLE.

Response
NonStop TM/MP is a critical resource for MQSeries. Immediately determine the cause using system utilities and rectify.

AMQ8647 (Tandem)
NonStop TM/MP reports transactions enabled

Severity
0 : Information

Explanation
The Queue Server <*insert_3*> has detected that the Compaq NonStop TM/MP transactions are enabled on the NSK system.

Response
No action is normally necessary. If transactions were previously disabled, this message indicates that the system has returned to normal operation.

AMQ8648 (Tandem)
A Queue Server has started

Severity
0 : Information

Explanation
A Queue Server in CPU <*insert_1*> has started. The process is named <*insert_3*>.

Response
None.

AMQ8649
Reset IBM WebSphere MQ Queue Manager accepted.

Severity
0 : Information

Explanation
The MQSC RESET QMGR command completed successfully. Details follow this message.

Response
None.

AMQ8650
Activity information unavailable.

Severity
0 : Information

Explanation
The DSPMQRTE command was expecting activity information but it was unavailable. This does not always constitute an error. Reasons why the activity information is unavailable include the following:

1) One of the queue managers on the route did not support trace-route messaging.

2) One of the queue managers on the route did not allow route information to be returned to the reply queue. See the documentation on the ActivityRecording and TraceRouteRecording queue manager attributes for more details.

3) The report could not a find route back to the reply queue.

Response

Try and determine whether the activity information should have been available. Running the command with the 'outline' verbosity option (used with the -v flag) may be useful in determining where the message was when the activity information was generated.

AMQ8650 (IBM i)

Activity information unavailable.

Severity

0 : Information

Explanation

The DSPMQMRTE command was expecting activity information but it was unavailable. This does not always constitute an error. Reasons why the activity information is unavailable include the following:

1) One of the queue managers on the route did not support trace-route messaging.

2) One of the queue managers on the route did not allow route information to be returned to the reply queue. See the documentation on the ActivityRecording and TraceRouteRecording queue manager attributes for more details.

3) The report could not a find route back to the reply queue.

Response

Try and determine whether the activity information should have been available. Running the command with DSPINF(*ALL) may be useful in determining where the message was when the activity information was generated.

AMQ8651

DSPMQRTE command has finished with errors.

Severity

0 : Information

Explanation

The DSPMQRTE command has finished processing your request but an execution error was detected. Previous messages issued by the command can be used to identify the error.

Response

Refer to previous messages issued by the command.

AMQ8651 (IBM i)

DSPMQMRTE command has finished with errors.

Severity

0 : Information

Explanation

The DSPMQMRTE command has finished processing your request but an execution error was detected. Previous messages issued by the command can be used to identify the error.

Response

Refer to previous messages issued by the command.

AMQ8652

DSPMQRTE command has finished.

Severity
0 : Information

Explanation
The DSPMQRTE command has finished processing your request and no execution errors were detected.

Response
None.

AMQ8652 (IBM i)
DSPMQMRTE command has finished.

Severity
0 : Information

Explanation
The DSPMQMRTE command has finished processing your request and no execution errors were detected.

Response
None.

AMQ8653
DSPMQRTE command started with options <*insert_3*>.

Severity
0 : Information

Explanation
You have started the DSPMQRTE command with command line options <*insert_3*> and the command is now processing your request.

Response
Wait for the command to finish processing your request. Any further messages that are issued can be used to determine the outcome of the request.

AMQ8653 (IBM i)
DSPMQMRTE command started.

Severity
0 : Information

Explanation
You have started the DSPMQMRTE command and the command is now processing your request.

Response
Wait for the command to finish processing your request. Any further messages that are issued can be used to determine the outcome of the request.

AMQ8654
Trace-route message arrived on queue manager <*insert_3*>.

Severity
0 : Information

Explanation
The DSPMQRTE command has received confirmation of the successful arrival of the trace-route message at its destination queue on queue manager <*insert_3*>.

Response
None.

AMQ8654 (IBM i)
Trace-route message arrived on queue manager <*insert_3*>.

Severity
0 : Information

Explanation
The DSPMQMRTE command has received confirmation of the successful arrival of the trace-route message at its destination queue on queue manager <*insert_3*>.

Response
None.

AMQ8655
Trace-route message expired.

Severity
0 : Information

Explanation
The DSPMQRTE command has received confirmation that the trace-route message has expired.

Response
The expiry interval of trace-route messages generated by the DSPMQRTE command can be altered using the -xs option if this is required.

AMQ8655 (IBM i)
Trace-route message expired.

Severity
0 : Information

Explanation
The DSPMQMRTE command has received confirmation that the trace-route message has expired.

Response
The expiry interval of trace-route messages generated by the DSPMQMRTE command can be altered using the EXPIRY parameter if this is required.

AMQ8656
DSPMQRTE command received an exception report from queue manager <*insert_4*> with feedback <*insert_1*> <*insert_3*>.

Severity
0 : Information

Explanation
The DSPMQRTE command trace-route message caused an exception on queue manager <*insert_4*>. The Feedback field in the report was <*insert_1*> or <*insert_3*>.

Response
Use the feedback given to determine why the trace-route message caused the exception.

AMQ8656 (IBM i)
DSPMQMRTE command received an exception report from queue manager <*insert_4*> with feedback <*insert_1*> <*insert_3*>.

Severity
0 : Information

Explanation
The DSPMQMRTE command trace-route message caused an exception on queue manager <*insert_4*>. The Feedback field in the report was <*insert_1*> or <*insert_3*>.

Response
Use the feedback given to determine why the trace-route message caused the exception.

AMQ8657
DSPMQRTE command used <*insert_3*> 0x<*insert_4*>.

Severity
0 : Information

Explanation
You started the DSPMQRTE command specifying that it should generate a trace-route message. This took place and the trace-route message had <*insert_3*> X<*insert_4*>.

Response
The <*insert_3*> can be used to retrieve responses to this trace-route request. Run the DSPMQRTE command again specifying this identifier with the -i flag and with the target queue specified as the queue where the responses are expected to return or where the trace-route message is expected to have arrived. This may be on another queue manager.

AMQ8657 (IBM i)
DSPMQMRTE command used <*insert_3*> 0x<*insert_4*>.

Severity
0 : Information

Explanation
You started the DSPMQMRTE command specifying that it should generate a trace-route message. This took place and the trace-route message had <*insert_3*> X<*insert_4*>.

Response
The <*insert_3*> can be used to retrieve responses to this trace-route request. Run the DSPMQMRTE command again specifying this identifier for CRLLID and with the target queue specified as the queue where the responses are expected to return or where the trace-route message is expected to have arrived. This may be on another queue manager.

AMQ8658
DSPMQRTE command failed to put a message to the specified target.

Severity
0 : Information

Explanation
The request for the DSPMQRTE command to put a trace-route message was unsuccessful. Previous messages issued by the command can be used to identify why the message could not be put.

Response
Refer to previous messages issued by the command.

AMQ8658 (IBM i)
DSPMQMRTE command failed to put a message on the target queue.

Severity
0 : Information

Explanation
The request for the DSPMQMRTE command to put a trace-route message on the target queue was unsuccessful. Previous messages issued by the command can be used to identify why the message could not be put on the target queue.

Response
Refer to previous messages issued by the command.

AMQ8659
DSPMQRTE command successfully put a message on queue <*insert_3*>, queue manager <*insert_4*>.

Severity
0 : Information

Explanation
The request for the DSPMQRTE command to put a message on the target queue was successful. The target queue resolved to <*insert_3*> on queue manager <*insert_4*>.

Response
None.

AMQ8659 (IBM i)
DSPMQMRTE command successfully put a message on queue <*insert_3*>, queue manager <*insert_4*>.

Severity
0 : Information

Explanation
The request for the DSPMQMRTE command to put a message on the target queue was successful. The target queue resolved to <*insert_3*> on queue manager <*insert_4*>.

Response
None.

AMQ8660
DSPMQRTE command could not correctly order the following activities:

Severity
0 : Information

Explanation
The DSPMQRTE command received the following activities, but they could not be printed in the correct order. This is commonly because an activity report has been received that does not contain a TraceRoute PCF group or is missing the RecordedActivities parameter which would allow it to be ordered correctly.

Response
Find and correct the application that is generating activity reports without the necessary information for them to be ordered correctly.

AMQ8660 (IBM i)
DSPMQMRTE command could not correctly order the following activities:

Severity
0 : Information

Explanation
The DSPMQMRTE command received the following activities, but they could not be printed in the correct order. This is commonly because an activity report has been received that does not contain a TraceRoute PCF group or is missing the RecordedActivities parameter which would allow it to be ordered correctly.

Response
Find and correct the application that is generating activity reports without the necessary information for them to be ordered correctly.

AMQ8661
DSPMQRTE command will not put to queue <*insert_3*>, queue manager <*insert_4*>.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying that the trace-route message must not be

delivered to a local queue (-d yes was not specified). However, it has been determined that the target queue does not resolve to a transmission queue. Therefore the DSPMQRTE command has chosen not to put the trace-route message to the target queue <*insert_3*> on queue manager <*insert_4*>.

Response
Determine whether it was expected that the target queue would resolve to a local queue.

AMQ8661 (IBM i)
DSPMQMRTE command will not put to queue <*insert_3*>, queue manager <*insert_4*>.

Severity
20 : Error

Explanation
You started the DSPMQMRTE command specifying that the trace-route message must not be delivered to a local queue (DLVRMSG(*NO) was specified). However, it has been determined that the target queue does not resolve to a transmission queue. Therefore the DSPMQMRTE command has chosen not to put the trace-route message to the target queue <*insert_3*> on queue manager <*insert_4*>.

Response
Determine whether it was expected that the target queue would resolve to a local queue.

AMQ8662
Trace-route message delivered on queue manager <*insert_3*>.

Severity
0 : Information

Explanation
The DSPMQRTE command has received confirmation of the successful delivery of the trace-route message on queue manager <*insert_3*> to a requesting application.

Response
None.

AMQ8662 (IBM i)
Trace-route message delivered on queue manager <*insert_3*>.

Severity
0 : Information

Explanation
The DSPMQMRTE command has received confirmation of the successful delivery of the trace-route message on queue manager <*insert_3*> to a requesting application.

Response
None.

AMQ8663
Client connection not supported in this environment.

Severity
20 : Error

Explanation
An attempt was made to connect to a queue manager using a client connection. However, client connections are not supported in your environment.

Response
Connect to the queue manager using a server connection.

AMQ8664
DSPMQRTE command could not connect to queue manager <*insert_3*>.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying that it should connect to queue manager <*insert_3*>. The command could not connect to that queue manager. Previous messages issued by the command can be used to identify the error.

Response
Refer to previous messages issued by the command.

AMQ8664 (IBM i)
DSPMQMRTE command could not connect to queue manager <*insert_3*>.

Severity
20 : Error

Explanation
You started the DSPMQMRTE command specifying that it should connect to queue manager <*insert_3*>. The command could not connect to that queue manager. Previous messages issued by the command can be used to identify the error.

Response
Refer to previous messages issued by the command.

AMQ8665
DSPMQRTE command was supplied an invalid CorrelId <*insert_3*>.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying option -i with a CorrelId <*insert_3*> that was invalid. The CorrelId was either too long or not in the correct format.

Response
Refer to the command syntax, and then try the command again.

AMQ8665 (IBM i)
DSPMQMRTE command was supplied an invalid CorrelId <*insert_3*>.

Severity
20 : Error

Explanation
You started the DSPMQMRTE command specifying CRLLID with a CorrelId <*insert_3*> that was invalid.

Response
Refer to the command syntax, and then try the command again.

AMQ8666
Queue <*insert_3*> on queue manager <*insert_4*>.

Severity
0 : Information

Explanation
The DSPMQRTE command trace-route message has been confirmed as having taken a route involving queue <*insert_3*> on queue manager <*insert_4*> in an attempt to reach the destination queue.

Response
Wait for subsequent messages which may indicate other queues or topics which the resultant message has been routed through.

AMQ8666 (IBM i)

Queue <*insert_3*> on queue manager <*insert_4*>.

Severity

0 : Information

Explanation

The DSPMQMRTE command trace-route message has been confirmed as having taken a route involving queue <*insert_3*> on queue manager <*insert_4*> in an attempt to reach the destination queue.

Response

Wait for subsequent messages which may indicate another queue which the message has been routed through.

AMQ8667

DSPMQRTE command could not open reply queue <*insert_3*>, queue manager <*insert_4*>.

Severity

20 : Error

Explanation

You started the DSPMQRTE command specifying reply queue <*insert_3*>. However the DSPMQRTE command could not successfully open a queue of that name on queue manager <*insert_4*>. Previous messages issued by the command can be used to identify the error. If the -rq option was not specified then the reply queue will be a temporary dynamic queue modelled on SYSTEM.DEFAULT.MODEL.QUEUE.

Response

Refer to previous messages issued by the command. Specify a reply queue that can be opened and then retry the command.

AMQ8667 (IBM i)

DSPMQMRTE command could not open reply queue <*insert_3*>, queue manager <*insert_4*>.

Severity

20 : Error

Explanation

You started the DSPMQMRTE command specifying reply queue <*insert_3*>. However the DSPMQMRTE command could not successfully open a queue of that name on queue manager <*insert_4*>. Previous messages issued by the command can be used to identify the error. If the RPLYQ parameter was not specified then the reply queue will be a temporary dynamic queue modelled on SYSTEM.DEFAULT.MODEL.QUEUE.

Response

Refer to previous messages issued by the command. Specify a reply queue that can be opened and then retry the command.

AMQ8668

DSPMQRTE command could not open queue <*insert_3*>, queue manager <*insert_4*>.

Severity

20 : Error

Explanation

You started the DSPMQRTE command specifying queue <*insert_3*>, using the -q option. However the DSPMQRTE command could not successfully open a queue of that name on queue manager <*insert_4*>. Previous messages issued by the command can be used to identify the error.

Response

Refer to previous messages issued by the command. Specify a queue, using the -q option, that can be opened and then retry the command.

AMQ8668 (IBM i)

DSPMQMRTE command could not open queue <*insert_3*>, queue manager <*insert_4*>.

Severity

20 : Error

Explanation

You started the DSPMQMRTE command specifying queue <*insert_3*> for the QNAME parameter. However the DSPMQMRTE command could not successfully open a queue of that name on queue manager <*insert_4*>. Previous messages issued by the command can be used to identify the error.

Response

Refer to previous messages issued by the command. Specify a queue, using the QNAME parameter, that can be opened and then retry the command.

AMQ8669

DSPMQRTE command failed to resolve queue manager <*insert_3*> on queue manager <*insert_4*>.

Severity

20 : Error

Explanation

The DSPMQRTE command attempted to resolve queue manager <*insert_3*> (supplied by the -qm option) on queue manager <*insert_4*> but the attempt failed. The queue specified by the -q option could not be opened.

Response

Ensure that queue manager <*insert_3*> can be resolved on queue manager <*insert_4*> or specify a different queue manager with the -qm option. Retry the command.

AMQ8669 (IBM i)

DSPMQMRTE command failed to resolve queue manager <*insert_3*> on queue manager <*insert_4*>.

Severity

20 : Error

Explanation

The DSPMQMRTE command attempted to resolve queue manager <*insert_3*> (supplied by the TGTMQM parameter) on queue manager <*insert_4*> but the attempt failed. The queue specified by the QNAME parameter could not be opened.

Response

Ensure that queue manager <*insert_3*> can be resolved on queue manager <*insert_4*> or specify a different queue manager with the TGTMQM parameter. Retry the command.

AMQ8670

Loading of server module <*insert_3*> failed.

Severity

20 : Error

Explanation

An attempt to dynamically load the server module <*insert_3*> failed. Typically this is because only the client modules are installed.

Response

Check which modules are installed and retry the command with the -c option specified if applicable.

AMQ8671

DSPMQRTE command was not supplied a reply queue when one was required.

Severity
20 : Error

Explanation
The DSPMQRTE command was expecting a reply queue specified by the -rq option but no reply queue was specified. Specifying a reply queue is mandatory if both the -n (no display) option and a response generating option (-ar or -ro [activity | coa | cod | exception | expiration]) is specified.

Response
Specify a reply queue and retry the command.

AMQ8672
DSPMQRTE command failed to get a message from queue <*insert_3*>, queue manager <*insert_4*>.

Severity
20 : Error

Explanation
The DSPMQRTE command attempted to get a message from queue <*insert_3*>, queue manager <*insert_4*>, but the attempt failed. Previous messages issued by the command can be used to identify the error.

Response
Refer to previous messages issued by the command.

AMQ8672 (IBM i)
DSPMQMRTE command failed to get a message from queue <*insert_3*>, queue manager <*insert_4*>.

Severity
20 : Error

Explanation
The DSPMQMRTE command attempted to get a message from queue <*insert_3*>, queue manager <*insert_4*>, but the attempt failed. Previous messages issued by the command can be used to identify the error.

Response
Refer to previous messages issued by the command.

AMQ8673
DSPMQRTE command was supplied option <*insert_3*> with an invalid object name <*insert_4*>.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying option <*insert_3*> with an object name <*insert_4*> that is invalid. In general, the names of IBM WebSphere MQ objects can have up to 48 characters. An object name can contain the following characters:

1) Uppercase alphabetic characters (A through Z).

2) Lowercase alphabetic characters (a through z).

3) Numeric digits (0 through 9).

4) Period (.), forward slash (/), underscore (_), percent (%).

See the IBM WebSphere MQ System Administration documentation for further details and restrictions.

Response
Specify a valid object name and then try the command again.

AMQ8673 (IBM i)
DSPMQMRTE command was supplied with an invalid object name <*insert_4*>.

Severity
20 : Error

Explanation
You started the DSPMQMRTE command specifying an object name <*insert_4*> that is invalid. In general, the names of IBM WebSphere MQ objects can have up to 48 characters. An object name can contain the following characters:

1) Uppercase alphabetic characters (A through Z).

2) Lowercase alphabetic characters (a through z).

3) Numeric digits (0 through 9).

4) Period (.), forward slash (/), underscore (_), percent (%).

See the IBM WebSphere MQ System Administration documentation for further details and restrictions.

Response
Specify a valid object name and then try the command again.

AMQ8674
DSPMQRTE command is now waiting for information to display.

Severity
0 : Information

Explanation
The DSPMQRTE command has successfully generated and put the trace-route message and is now waiting for responses to be returned to the reply queue to indicate the route that the trace-route message took to its destination.

Response
Wait for responses to be returned to the reply queue and for the information about the route to be displayed.

AMQ8674 (IBM i)
DSPMQMRTE command is now waiting for information to display.

Severity
0 : Information

Explanation
The DSPMQMRTE command has successfully generated and put the trace-route message and is now waiting for responses to be returned to the reply queue to indicate the route that the trace-route message took to its destination.

Response
Wait for responses to be returned to the reply queue and for the information about the route to be displayed.

AMQ8675
DSPMQRTE command was supplied an invalid option <*insert_3*>.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying an option of <*insert_3*> that was not recognized. The command will end.

Response
Refer to the command syntax and retry the command.

AMQ8676
DSPMQRTE command was supplied an invalid combination of options.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying a combination of the options that is not valid. Only one of -ts or -q must be specified. The -i option cannot be specified with one or more of the following options: -ac, -ar, -d, -f, -l, -n, -o, -p, -qm, -ro, -rq, -rqm, -s, -t, -xs, -xp. The -n option cannot be specified with one or more of the following options: -b, -i, -v, -w. The -ar option can only be specified if the -ac option has also been specified. The -rqm option can only be specified if the -rq option has also been specified.

Response
Refer to the command documentation and then try the command again.

AMQ8677
DSPMQRTE command was supplied an option <*insert_3*> with conflicting values.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying values for option <*insert_3*> that conflict. At least two values were specified for the same option but they conflict with each other. The DSPMQRTE command will end.

Response
Refer to the command syntax and then try the command again.

AMQ8677 (IBM i)
DSPMQMRTE command was supplied a parameter with conflicting values.

Severity
20 : Error

Explanation
You started the DSPMQMRTE command specifying values that conflict. At least two values were specified for the same parameter but they conflict with each other. The DSPMQMRTE command will end.

Response
Refer to the command syntax and then try the command again.

AMQ8678
DSPMQRTE command was supplied option <*insert_3*> with an invalid value <*insert_4*>.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying an invalid option value. The <*insert_4*> value for option <*insert_3*> is either not recognized or of an incorrect format.

Response
Refer to the command syntax, and then try the command again.

AMQ8678 (IBM i)
DSPMQMRTE command was supplied an invalid value <*insert_4*>.

Severity
20 : Error

Explanation
You started the DSPMQMRTE command specifying an invalid parameter value. Value <*insert_4*> is either not recognized or of an incorrect format.

Response
Refer to the command syntax, and then try the command again.

AMQ8679
Persistent messages not allowed on reply queue <*insert_3*>, queue manager <*insert_4*>.

Severity
20 : Error

Explanation
It was specified that the DSPMQRTE command should put a persistent trace-route message on the target queue (see the documentation for the -l option). However, persistent messages are not allowed on the reply queue because it is a temporary dynamic queue and persistent responses were expected to return to it. The trace-route message was not put on the target queue.

Response
Ensure that the reply queue is not a temporary dynamic queue. Use the -rq option to specify the reply queue.

AMQ8679 (IBM i)
Persistent messages not allowed on reply queue <*insert_3*>, queue manager <*insert_4*>.

Severity
20 : Error

Explanation
It was specified that the DSPMQMRTE command should put a persistent trace-route message on the target queue (see the documentation for the MSGPST parameter). However, persistent messages are not allowed on the reply queue because it is a temporary dynamic queue and persistent responses were expected to return to it. The trace-route message was not put on the target queue.

Response
Ensure that the reply queue is not a temporary dynamic queue. Use the RPLYQ parameter to specify the reply queue.

AMQ8680
DSPMQRTE command failed to open queue manager <*insert_3*>.

Severity
20 : Error

Explanation
The DSPMQRTE command tried to open queue manager <*insert_3*> for inquire but the open failed. Previous messages issued by the command can be used to identify the error.

Response
Refer to previous messages issued by the command.

AMQ8680 (IBM i)
DSPMQMRTE command failed to open queue manager <*insert_3*>.

Severity
20 : Error

Explanation
The DSPMQMRTE command tried to open queue manager <*insert_3*> for inquire but the open failed. Previous messages issued by the command can be used to identify the error.

Response
Refer to previous messages issued by the command.

AMQ8681
DSPMQRTE command has detected an error, reason <*insert_1*> <*insert_3*>.

Severity
20 : Error

Explanation
The DSPMQRTE command has detected an error from an MQI call during the execution of your request. The reason for failure is <*insert_1*> or <*insert_3*>.

Response
See the IBM WebSphere MQ Messages documentation for an explanation of the reason for failure. Follow any correction action and retry the command.

AMQ8681 (IBM i)
DSPMQMRTE command has detected an error, reason <*insert_1*> <*insert_3*>.

Severity
20 : Error

Explanation
The DSPMQMRTE command has detected an error from an MQI call during the execution of your request. The reason for failure is <*insert_1*> or <*insert_3*>.

Response
See the IBM WebSphere MQ Messages documentation for an explanation of the reason for failure. Follow any correction action and retry the command.

AMQ8682
Trace-route message processed by application <*insert_3*> on queue manager <*insert_4*>.

Severity
0 : Information

Explanation
The DSPMQRTE command successfully put a trace-route message on the target queue and it was then delivered by queue manager <*insert_4*> to application <*insert_3*> which processed the message.

Response
Determine if it was expected that this application would process the trace-route message.

AMQ8682 (IBM i)
Trace-route message processed by application <*insert_3*> on queue manager <*insert_4*>.

Severity
0 : Information

Explanation
The DSPMQMRTE command successfully put a trace-route message on the target queue and it was then delivered by queue manager <*insert_4*> to application <*insert_3*> which processed the message.

Response
Determine if it was expected that this application would process the trace-route message.

AMQ8683
Trace-route message reached the maximum activities limit of <*insert_1*>.

Severity
0 : Information

Explanation
The DSPMQRTE command trace-route message was rejected after the number of activities of which it was a participant reached the maximum activities limit. The limit was set to <*insert_1*>. The maximum activities limit is set using the -s option.

Response
Using the output from the command determine whether it is expected that the trace-route message should have reached the maximum activities limit.

AMQ8683 (IBM i)
Trace-route message reached the maximum activities limit of <*insert_1*>.

Severity
0 : Information

Explanation
The DSPMQMRTE command trace-route message was rejected after the number of activities of which it was a participant reached the maximum activities limit. The limit was set to <*insert_1*>. The maximum activities limit is set using the MAXACTS parameter.

Response
Using the output from the command determine whether it is expected that the trace-route message should have reached the maximum activities limit.

AMQ8684
Trace-route message reached trace-route incapable queue manager <*insert_3*>.

Severity
0 : Information

Explanation
The DSPMQRTE command trace-route message was rejected because it was about to be sent to a queue manager which does not support trace-route messaging. This behaviour was requested because the forwarding options specified on the command only allowed the trace-route message to be forwarded to queue managers which support trace-route messaging. Sending a trace-route message to a queue manager which cannot process it in accordance with its specified options could cause undesirable results, including having the trace-route message be put to a local queue on the remote queue manager. If this is acceptable then the '-f all' option can be specified.

Response
Retry the command with different forwarding options, if appropriate.

AMQ8684 (IBM i)
Trace-route message reached trace-route incapable queue manager <*insert_3*>.

Severity
0 : Information

Explanation
The DSPMQMRTE command trace-route message was rejected because it was about to be sent to a queue manager which does not support trace-route messaging. This behaviour was requested because the forwarding options specified on the command only allowed the trace-route message to be forwarded to queue managers which support trace-route messaging. Sending a trace-route message to a queue manager which cannot process it in accordance with its specified options could cause undesirable results, including having the trace-route message be put to a local queue on the remote queue manager. If this is acceptable then FWDMSG(*ALL) can be specified.

Response
Retry the command with different forwarding options, if appropriate.

AMQ8685
Trace-route message rejected due to invalid forwarding options X<*insert_1*>.

Severity
20 : Error

Explanation
The DSPMQRTE command trace-route message was rejected because one or more of the forwarding options was not recognized and it was in the MQROUTE_FORWARD_REJ_UNSUP_MASK bitmask. The forwarding options, when they were last observed, in hexadecimal were X<*insert_1*>.

Response
Change the application that inserted the forwarding options that were not recognized to insert valid and supported forwarding options.

AMQ8685 (IBM i)
Trace-route message rejected due to invalid forwarding options X<*insert_1*>.

Severity
20 : Error

Explanation
The DSPMQMRTE command trace-route message was rejected because one or more of the forwarding options was not recognized and it was in the MQROUTE_FORWARD_REJ_UNSUP_MASK bitmask. The forwarding options, when they were last observed, in hexadecimal were X<*insert_1*>.

Response
Change the application that inserted the forwarding options that were not recognized to insert valid and supported forwarding options.

AMQ8686
Trace-route message rejected due to invalid delivery options X<*insert_1*>.

Severity
20 : Error

Explanation
The DSPMQRTE command trace-route message was rejected because one or more of the delivery options was not recognized and it was in the MQROUTE_DELIVER_REJ_UNSUP_MASK bitmask. The delivery options, when they were last observed, in hexadecimal were X<*insert_1*>.

Response
Change the application that inserted the delivery options that were not recognized to insert valid and supported delivery options.

AMQ8686 (IBM i)
Trace-route message rejected due to invalid delivery options X<*insert_1*>.

Severity
20 : Error

Explanation
The DSPMQMRTE command trace-route message was rejected because one or more of the delivery options was not recognized and it was in the MQROUTE_DELIVER_REJ_UNSUP_MASK bitmask. The delivery options, when they were last observed, in hexadecimal were X<*insert_1*>.

Response
Change the application that inserted the delivery options that were not recognized to insert valid and supported delivery options.

AMQ8687
Program ending.

Severity
0 : Information

Explanation
The program operation was interrupted by a SIGINT signal on UNIX systems or a CTRL+c/CTRL+BREAK signal on Windows systems. The program is now ending.

Response
Wait for the program to end.

AMQ8688
DSPMQRTE command has detected an unexpected error, reason <*insert_1*> <*insert_3*>.

Severity
20 : Error

Explanation
The DSPMQRTE command has detected an unexpected error during execution of your request. The reason for failure is <*insert_1*> or <*insert_3*>. The IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8688 (IBM i)
DSPMQMRTE command has detected an unexpected error, reason <*insert_1*> <*insert_3*>.

Severity
20 : Error

Explanation
The DSPMQMRTE command has detected an unexpected error during execution of your request. The reason for failure is <*insert_1*> or <*insert_3*>. The IBM WebSphere MQ error recording routine has been called.

Response
Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8689
Loading of client module <*insert_3*> failed.

Severity
20 : Error

Explanation
An attempt to dynamically load the client module <*insert_3*> failed. Typically this is because the client modules are not installed.

Response
Check which modules are installed and retry the command without the -c option specified, if applicable.

AMQ8690
IBM WebSphere MQ topic created.

Severity
0 : Information

Explanation
IBM WebSphere MQ topic <*insert_3*> created.

Response
None.

AMQ8691
IBM WebSphere MQ topic changed.

Severity
0 : Information

Explanation
IBM WebSphere MQ topic <*insert_5*> changed.

Response
None.

AMQ8692
IBM WebSphere MQ topic object deleted.

Severity
0 : Information

Explanation
IBM WebSphere MQ topic object <*insert_3*> deleted.

Response
None.

AMQ8694
DSPMQRTE command successfully put a message to topic string <*insert_3*>, queue manager <*insert_4*>.

Severity
0 : Information

Explanation
The request for the DSPMQRTE command to put a message was successful. The destination specified resolved to topic string <*insert_3*> on queue manager <*insert_4*>.

Response
None.

AMQ8695
Topic string <*insert_3*> on queue manager <*insert_4*>.

Severity
0 : Information

Explanation
The DSPMQRTE command trace-route message has been confirmed as having taken a route involving topic string <*insert_3*> on queue manager <*insert_4*>.

Response
Wait for subsequent messages which may indicate other queues or topics which the resultant messages have been routed through.

AMQ8696
DSPMQRTE command could not open topic string <*insert_3*>, queue manager <*insert_4*>.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying topic string <*insert_3*>, using the -ts option. However the DSPMQRTE command could not successfully open that topic string on queue manager <*insert_4*>. Previous messages issued by the command can be used to identify the error.

Response
Refer to previous messages issued by the command. Specify a topic string, using the -ts option, that can be opened and then retry the command.

AMQ8697
DSPMQRTE command could not open topic <*insert_3*>, queue manager <*insert_4*>.

Severity
20 : Error

Explanation
You started the DSPMQRTE command specifying topic <*insert_3*>, using the -to option. However the DSPMQRTE command could not successfully open a topic object of that name on queue manager <*insert_4*>. Previous messages issued by the command can be used to identify the error.

Response
Refer to previous messages issued by the command. Specify a topic, using the -to option, that can be opened and then retry the command.

AMQ8698
Too many keywords have been specified.

Severity
0 : Information

Explanation
Too many keywords for the command have been specified.

Response
None

AMQ8701
Usage: rcdmqimg [-z] [-l] [-m QMgrName] -t ObjType [GenericObjName]

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8702
Usage: rcrmqobj [-z] [-m QMgrName] -t ObjType [GenericObjName]

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8703
Usage: dspmqfls [-m QMgrName] [-t ObjType] GenericObjName

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8704 (Tandem)
Usage: altmqfls [--qmgr QMgrName] [--type ObjType] [--volume Volume] [-server ServerName] [--qsoptions options] [--msgofthresh Threshold] [--browse Bytes] [--meascount counter] [--qsize (primaryextent,secondaryextent, maxextents)] [--oflowsize (primaryextent,secondaryextent, maxextents)] ObjectName

Severity
0 : Information

Response
None.

AMQ8705
Display Queue Manager Status Details.

Severity
0 : Information

Explanation
The MQSC DISPLAY QMSTATUS command completed successfully. Details follow this message.

Response
None.

AMQ8706
Request to stop IBM WebSphere MQ Listener accepted.

Severity
0 : Information

Explanation
The channel listener program has been requested to stop. This command executes asynchronously so may complete after this message has been displayed.

Response
Further information on the progress of the request is available in the queue manager error log.

AMQ8707 (IBM i)
Start IBM WebSphere MQ DLQ Handler

Severity
0 : Information

AMQ8708
Dead-letter queue handler started to process INPUTQ(<*insert_3*>).

Severity
0 : Information

Explanation
The dead-letter queue handler (runmqdlq) has been started and has parsed the input file without detecting any errors and is about to start processing the queue identified in the message.

Response
None.

AMQ8708 (IBM i)
Dead-letter queue handler started to process INPUTQ(<*insert_3*>).

Severity
0 : Information

Explanation
The dead-letter queue handler (STRMQMDLQ) has been started and has parsed the input file without detecting any errors and is about to start processing the queue identified in the message.

Response
None.

AMQ8709
Dead-letter queue handler ending.

Severity
0 : Information

Explanation
The dead-letter queue handler (runmqdlq) is ending because the WAIT interval has expired and there are no messages on the dead-letter queue, or because the queue manager is shutting down, or because the dead-letter queue handler has detected an error. If the dead-letter queue handler has detected an error, an earlier message will have identified the error.

Response
None.

AMQ8709 (IBM i)
Dead-letter queue handler ending.

Severity
0 : Information

Explanation
The dead-letter queue handler (STRMQMDLQ) is ending because the WAIT interval has expired and there are no messages on the dead-letter queue, or because the queue manager is shutting down, or because the dead-letter queue handler has detected an error. If the dead-letter queue handler has detected an error, an earlier message will have identified the error.

Response
None.

AMQ8710
Usage: runmqdlq [QName[QMgrName]].

Severity
0 : Information

Explanation
Syntax for the usage of runmqdlq.

Response
None.

AMQ8711 (IBM i)
Job <*insert_3*> has terminated unexpectedly.

Severity
10 : Warning

Explanation
Execution of the command <*insert_5*> caused job <*insert_3*> to be started, but the job terminated unexpectedly.

Response
Consult the log for job <*insert_3*> to determine why it was terminated.

AMQ8712
PubSub is disabled for this queue manager.

Severity
40 : Stop Error

Explanation
The queue manager configuration inhibits any publication or subscription commands.

Response
Check the queue manager options and ensure they are correct.

AMQ8721
Dead-letter queue message not prefixed by a valid MQDLH.

Severity
10 : Warning

Explanation
The dead-letter queue handler (runmqdlq) retrieved a message from the nominated dead-letter queue, but the message was not prefixed by a recognizable MQDLH. This typically occurs because an application is writing directly to the dead-letter queue but is not prefixing messages with a valid MQDLH. The message is left on the dead-letter queue and the dead-letter queue handler continues to process the dead-letter queue. Each time the dead-letter queue handler repositions itself to a position before this message to process messages that could not be processed on a previous scan it will reprocess the failing message and will consequently re-issue this message.

Response
Remove the invalid message from the dead-letter queue. Do not write messages to the dead-letter queue unless they have been prefixed by a valid MQDLH. If you require a dead-letter queue handler that can process messages not prefixed by a valid MQDLH, you must change the sample program called amqsdlq to cater for your needs.

AMQ8721 (IBM i)
Dead-letter queue message not prefixed by a valid MQDLH.

Severity
10 : Warning

Explanation
The dead-letter queue handler (STRMQMDLQ) retrieved a message from the nominated dead-letter queue, but the message was not prefixed by a recognizable MQDLH. This typically occurs because an application is writing directly to the dead-letter queue but is not prefixing messages with a valid MQDLH. The message is left on the dead-letter queue and the dead-letter queue handler continues to process the dead-letter queue. Each time the dead-letter queue handler repositions itself to a position before this message to process messages that could not be processed on a previous scan it will reprocess the failing message and will consequently re-issue this message.

Response
Remove the invalid message from the dead-letter queue. Do not write messages to the dead-letter queue unless they have been prefixed by a valid MQDLH. If you require a dead-letter queue handler that can process messages not prefixed by a valid MQDLH, you must change the sample program called amqsdlq to cater for your needs.

AMQ8722
Dead-letter queue handler unable to put message: Rule <*insert_1*> Reason <*insert_2*>.

Severity
10 : Warning

Explanation
This message is produced by the dead-letter queue handler when it is requested to redirect a message to another queue but is unable to do so. If the reason that the redirect fails is the same as the reason the message was put to the dead-letter queue then it is assumed that no new error has occurred and no message is produced. The retry count for the message will be incremented and the dead-letter queue handler will continue.

Response
Investigate why the dead-letter queue handler was unable to put the message to the dead-letter queue. The line number of the rule used to determine the action for the message should be used to help identify to which queue the dead-letter queue handler attempted to PUT the message.

AMQ8723
Display pub/sub status details.

Severity
0 : Information

Explanation
The MQSC DISPLAY PUBSUB command completed successfully. Details follow this message.

AMQ8724
Refresh IBM WebSphere MQ Queue Manager accepted.

Severity
0 : Information

Explanation
The MQSC REFRESH QMGR command completed successfully. Details follow this message.

Response
None.

AMQ8729
The listener could not be stopped at this time.

Severity
10 : Warning

Explanation
A request was made to stop a listener, however the listener could not be stopped at this time. Reasons why a listener could not be stopped are:

The listener has active channels and the communications protocol being used is LU 6.2, SPX or NETBIOS.

The listener has active channels and the communications protocol being used is TCP/IP and channel threads are restricted to run within the listener process.

Response
End the channels using the STOP CHANNEL command and reissue the request.

AMQ8730
Listener already active.

Severity
10 : Warning

Explanation
A request was made to start a listener, however the listener is already running and cannot be started.

Response
If you do not want the listener to be running, then use the STOP LISTENER command to stop the listener before reissuing the command.

AMQ8731
Listener not active.

Severity
10 : Warning

Explanation
A request was made to stop a listener, however the listener is not running.

Response
If the listener should be running then use the START LISTENER command to start the listener.

AMQ8732
Request to stop Service accepted.

Severity
0 : Information

Explanation
The Request to stop the Service has been accepted and is being processed.

Response
None.

AMQ8733
Request to start Service accepted.

Severity
0 : Information

Explanation
The Request to start the Service has been accepted and is being processed.

Response
None.

AMQ8734
Command failed - Program could not be started.

Severity
20 : Error

Explanation
The command requested was unsuccessful because the program which was defined to be run to complete the action could not be started.

Reasons why the program could not be started are

The program does not exist at the specified location.

The WebSphere MQ user does not have sufficient access to execute the program.

If STDOUT or STDERR are defined for the program, the IBM WebSphere MQ user does not have sufficient access to the locations specified.

Response
Check the Queue Manager error logs for further details on the cause of the failure and correct before reissuing the command.

AMQ8735
Command failed - Access denied.

Severity
20 : Error

Explanation
The command requested was unsuccessful because access was denied attempting to execuete the program defined to run.

Response
Examine the definition of the object and ensure that the path to program file is correct. If the defined path is correct ensure that the program exists at the location specified and that the WebSphere MQ user has access to execute the program.

AMQ8737
Service already active.

Severity
10 : Warning

Explanation
A request was made to start a service, however the service is already running and cannot be started.

Response
If you do not want the service to be running, then use the STOP SERVICE command to stop the service before reissuing the command. If the intention is to allow more than one instance of s service to run, then the service definition may be altered to be of SERVTYPE(COMMAND) which allows more than one instance of the service to be executed concurrently, however status of services of type COMMAND is not available from the SVSTAUS command.

AMQ8738
Service not active.

Severity
10 : Warning

Explanation
A request was made to stop a service, however the service is not running.

Response
If the service should be running then use the START SERVICE command to start the service.

AMQ8739
Stop cannot be executed for service with blank STOPCMD.

Severity
20 : Error

Explanation
A request was made to STOP a service, however the service has no Stop Command defined so no action could be taken.

Response
Examine the definition of the service and if necessary update the definition of the service to include the command to run when STOP is issued. For services of type 'SERVER' the command to run when STOP is executed is stored when the service is started so any alteration to the service definition will have no effect until the service is restarted following the update.

AMQ8740
Start cannot be executed for service with blank STARTCMD.

Severity
20 : Error

Explanation

A request was made to START a service, however the service has no Start Command defined so no action could be taken.

Response

Examine the definition of the service and if necessary update the definition of the service to include the command to run when START is issued.

AMQ8741

Unable to connect to queue manager.

Severity

20 : Error

Explanation

The dead-letter queue handler (runmqdlq) could not connect to queue manager <*insert_3*>. This message is typically issued when the requested queue manager has not been started or is quiescing, or if the process does not have sufficient authority. The completion code (<*insert_1*>) and the reason (<*insert_2*>) can be used to identify the error. The dead-letter queue handler ends.

Response

Take appropriate action based upon the completion code and reason.

AMQ8741 (IBM i)

Unable to connect to queue manager.

Severity

20 : Error

Explanation

The dead-letter queue handler (STRMQMDLQ) could not connect to queue manager <*insert_3*>. This message is typically issued when the requested queue manager has not been started or is quiescing, or if the process does not have sufficient authority. The completion code (<*insert_1*>) and the reason (<*insert_2*>) can be used to identify the error. The dead-letter queue handler ends.

Response

Take appropriate action based upon the completion code and reason.

AMQ8742

Unable to open queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

20 : Error

Explanation

The dead-letter queue handler (runmqdlq) could not open the queue manager object. This message is typically issued because of a resource shortage or because the process does not have sufficient authority. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Take appropriate action based upon the completion code and reason.

AMQ8742 (IBM i)

Unable to open queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

20 : Error

Explanation

The dead-letter queue handler (STRMQMDLQ) could not open the queue manager object. This message is typically issued because of a resource shortage or because the process does not have sufficient authority. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8743
Unable to inquire on queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (runmqdlq) could not inquire on the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8743 (IBM i)
Unable to inquire on queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (STRMQMDLQ) could not inquire on the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8744
Unable to close queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (runmqdlq) could not close the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8744 (IBM i)
Unable to close queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (STRMQMDLQ) could not close the queue manager. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8745
Unable to open dead-letter queue for browse.

Severity
20 : Error

Explanation
The dead-letter queue handler (runmqdlq) could not open the dead-letter queue <*insert_3*> for browsing. This message is typically issued because another process has opened the dead-letter queue for exclusive access, or because an invalid dead-letter queue name was specified. Other possible reasons include resource shortages or insufficient authority. The completion code(<*insert_1*>) and the reason(<*insert_2*>) can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8745 (IBM i)
Unable to open dead-letter queue for browse.

Severity
20 : Error

Explanation
The dead-letter queue handler (STRMQMDLQ) could not open the dead-letter queue <*insert_3*> for browsing. This message is typically issued because another process has opened the dead-letter queue for exclusive access, or because an invalid dead-letter queue name was specified. Other possible reasons include resource shortages or insufficient authority. The completion code(<*insert_1*>) and the reason(<*insert_2*>) can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8746
Unable to close dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (runmqdlq) could not close the dead-letter queue. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8746 (IBM i)
Unable to close dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (STRMQMDLQ) could not close the dead-letter queue. This message is typically issued because of a resource shortage or because the queue manager is ending. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8747
Integer parameter outside permissible range.

Severity
20 : Error

Explanation
The integer parameter (<*insert_2*>) supplied to the dead-letter handler was outside of the valid range for <*insert_3*> on line <*insert_1*>.

Response
Correct the input data and restart the dead-letter queue handler.

AMQ8748
Unable to get message from dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (runmqdlq) could not get the next message from the dead-letter queue. This message is typically issued because of the queue manager ending, a resource problem, or another process having deleted the dead-letter queue. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8748 (IBM i)
Unable to get message from dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (STRMQMDLQ) could not get the next message from the dead-letter queue. This message is typically issued because of the queue manager ending, a resource problem, or another process having deleted the dead-letter queue. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8749
Unable to commit/backout action on dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
20 : Error

Explanation
The dead-letter queue handler (runmqdlq) was unable to commit or backout an update to the dead-letter queue. This message is typically issued because of the queue manager ending, or because of a resource shortage. If the queue manager has ended, the update to the dead-letter queue (and any associated updates) will be backed out when the queue manager restarts. If the problem was due to a resource problem then the updates will be backed out when the dead-letter queue handler terminates. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Take appropriate action based upon the completion code and reason.

AMQ8749 (IBM i)

Unable to commit/backout action on dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

20 : Error

Explanation

The dead-letter queue handler (STRMQMDLQ) was unable to commit or backout an update to the dead-letter queue. This message is typically issued because of the queue manager ending, or because of a resource shortage. If the queue manager has ended, the update to the dead-letter queue (and any associated updates) will be backed out when the queue manager restarts. If the problem was due to a resource problem then the updates will be backed out when the dead-letter queue handler terminates. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Take appropriate action based upon the completion code and reason.

AMQ8750

No valid input provided to runmqdlq.

Severity

20 : Error

Explanation

Either no input was provided to runmqdlq, or the input to runmqdlq contained no valid message templates. If input was provided to runmqdlq but was found to be invalid, earlier messages will have been produced explaining the cause of the error. The dead-letter queue handler will ends.

Response

Correct the input data and restart the dead-letter queue handler.

AMQ8750 (IBM i)

No valid input provided to STRMQMDLQ.

Severity

20 : Error

Explanation

Either no input was provided to STRMQMDLQ, or the input to STRMQMDLQ contained no valid message templates. If input was provided to STRMQMDLQ but was found to be invalid, earlier messages will have been produced explaining the cause of the error. The dead-letter queue handler will ends.

Response

Correct the input data and restart the dead-letter queue handler.

AMQ8751

Unable to obtain private storage.

Severity

20 : Error

Explanation

The dead-letter queue handler (runmqdlq) was unable to obtain private storage. This problem would typically arise as a result of some more global problem. For example if there is a persistent problem that is causing messages to be written to the DLQ and the same problem (for example queue full) is preventing the dead-letter queue handler from taking the requested action with the message, it is necessary for the dead-letter queue handler to maintain a large amount of state data to remember the retry counts associated with each message, or if the dead-letter queue contains a large number of messages and the rules table has directed the dead-letter queue handler to ignore the messages.

Response

Investigate if some more global problem exists, and if the dead-letter queue contains a large number of messages. If the problem persists save any generated output files and use either the IBM WebSphere MQ support web page, or the IBM support assistant at the IBM SupportAssistant web page, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8751 (IBM i)

Unable to obtain private storage.

Severity

20 : Error

Explanation

The dead-letter queue handler (STRMQMDLQ) was unable to obtain private storage. This problem would typically arise as a result of some more global problem. For example if there is a persistent problem that is causing messages to be written to the DLQ and the same problem (for example queue full) is preventing the dead-letter queue handler from taking the requested action with the message, it is necessary for the dead-letter queue handler to maintain a large amount of state data to remember the retry counts associated with each message, or if the dead-letter queue contains a large number of messages and the rules table has directed the dead-letter queue handler to ignore the messages.

Response

Investigate if some more global problem exists, and if the dead-letter queue contains a large number of messages. If the problem persists save any generated output files and use either the IBM WebSphere MQ support web page, or the IBM support assistant at the IBM SupportAssistant web page, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8752

Parameter(<*insert_3*>) exceeds maximum length on line <*insert_1*>.

Severity

20 : Error

Explanation

A parameter supplied as input to the dead-letter handler exceeded the maximum length for parameters of that type.

Response

Correct the input data and restart the dead-letter queue handler.

AMQ8753

Duplicate parameter(<*insert_3*>) found on line <*insert_1*>.

Severity

20 : Error

Explanation

Two or more parameters of the same type were supplied on a single input line to the dead-letter queue handler.

Response

Correct the input and restart the dead-letter queue handler.

AMQ8754

Display topic status details.

Severity

0 : Information

Explanation
The MQSC DISPLAY TPSTATUS command completed successfully. Details follow this message.

AMQ8755
IBM WebSphere MQ topicstr cleared successfully.

Severity
0 : Information

Explanation
All messages on topicstr have been deleted.

AMQ8756
Error detected releasing private storage.

Severity
20 : Error

Explanation
The dead-letter queue handler (runmqdlq) was informed of an error while attempting to release an area of private storage. The dead-letter queue handler ends.

Response
This message should be preceded by a message or FFST information from the internal routine that detected the error. Take the action associated with the earlier error information.

AMQ8756 (IBM i)
Error detected releasing private storage.

Severity
20 : Error

Explanation
The dead-letter queue handler (STRMQMDLQ) was informed of an error while attempting to release an area of private storage. The dead-letter queue handler ends.

Response
This message should be preceded by a message or FFST information from the internal routine that detected the error. Take the action associated with the earlier error information.

AMQ8757
Integer parameter(<*insert_3*>) outside permissible range on line <*insert_1*>.

Severity
20 : Error

Explanation
An integer supplied as input to the dead-letter handler was outside of the valid range of integers supported by the dead-letter queue handler.

Response
Correct the input data and restart the dead-letter queue handler.

AMQ8758
<*insert_1*> errors detected in input to runmqdlq.

Severity
20 : Error

Explanation
One or more errors have been detected in the input to the dead-letter queue handler(runmqdlq). Error messages will have been generated for each of these errors. The dead-letter queue handler ends.

Response
Correct the input data and restart the dead-letter queue handler.

AMQ8758 (IBM i)
<*insert_1*> errors detected in input to STRMQMDLQ.

Severity
20 : Error

Explanation
One or more errors have been detected in the input to the dead-letter queue handler(STRMQMDLQ). Error messages will have been generated for each of these errors. The dead-letter queue handler ends.

Response
Correct the input data and restart the dead-letter queue handler.

AMQ8759
Invalid combination of parameters to dead-letter queue handler on line <*insert_1*>.

Severity
20 : Error

Explanation
An invalid combination of input parameters has been supplied to the dead-letter queue handler. Possible causes are: no ACTION specified, ACTION(FWD) but no FWDQ specified, HEADER(YES|NO) specified without ACTION(FWD).

Response
Correct the input data and restart the dead-letter queue handler.

AMQ8760
Unexpected failure while initializing process: Reason = <*insert_1*>.

Severity
30 : Severe error

Explanation
The dead-letter queue handler (runmqdlq) could not perform basic initialization required to use MQ services because of an unforeseen error. The dead-letter queue handler ends.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8760 (IBM i)
Unexpected failure while initializing process: Reason = <*insert_1*>.

Severity
30 : Severe error

Explanation
The dead-letter queue handler (STRMQMDLQ) could not perform basic initialization required to use MQ services because of an unforeseen error. The dead-letter queue handler ends.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a

solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8761

Unexpected failure while connecting to queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (runmqdlq) could not connect to the requested queue manager because of an unforeseen error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8761 (IBM i)

Unexpected failure while connecting to queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (STRMQMDLQ) could not connect to the requested queue manager because of an unforeseen error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8762

Unexpected error while attempting to open queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (runmqdlq) could not open the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8762 (IBM i)

Unexpected error while attempting to open queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (STRMQMDLQ) could not open the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8763

Unexpected error while inquiring on queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead letter queue handler (runmqdlq) could not inquire on the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8763 (IBM i)

Unexpected error while inquiring on queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead letter queue handler (STRMQMDLQ) could not inquire on the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8764

Unexpected error while attempting to close queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (runmqdlq) could not close the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8764 (IBM i)

Unexpected error while attempting to close queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (STRMQMDLQ) could not close the queue manager because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8765

Unexpected failure while opening dead-letter queue for browse: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (runmqdlq) could not open the dead-letter queue for browsing because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8765 (IBM i)

Unexpected failure while opening dead-letter queue for browse: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (STRMQMDLQ) could not open the dead-letter queue for browsing because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8766

Unexpected error while closing dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (runmqdlq) could not close the dead-letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8766 (IBM i)

Unexpected error while closing dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (STRMQMDLQ) could not close the dead-letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8767

Unexpected error while getting message from dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
30 : Severe error

Explanation
The dead-letter queue handler (runmqdlq) could not get the next message from the dead-letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8767 (IBM i)
Unexpected error while getting message from dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
30 : Severe error

Explanation
The dead-letter queue handler (STRMQMDLQ) could not get the next message from the dead-letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8768
Unexpected error committing/backing out action on dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
30 : Severe error

Explanation
The dead-letter queue handler (runmqdlq) was unable to either commit or backout an update to the dead-letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response
Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8768 (IBM i)
Unexpected error committing/backing out action on dead-letter queue: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity
30 : Severe error

Explanation

The dead-letter queue handler (STRMQMDLQ) was unable to either commit or backout an update to the dead-letter queue because of an unforeseen error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8769

Unable to disconnect from queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (runmqdlq) was unable to disconnect from the queue manager because of an unexpected error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8769 (IBM i)

Unable to disconnect from queue manager: CompCode = <*insert_1*> Reason = <*insert_2*>.

Severity

30 : Severe error

Explanation

The dead-letter queue handler (STRMQMDLQ) was unable to disconnect from the queue manager because of an unexpected error. The completion code and the reason can be used to identify the error. The dead-letter queue handler ends.

Response

Use the standard facilities supplied with your system to record the problem identifier and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8770 (IBM)

Cannot open <*insert_3*> for command <*insert_5*>.

Severity

40 : Stop Error

Explanation

The <*insert_5*> command failed to open <*insert_3*> for IBM WebSphere MQ processing.

Response
Check that the intended file or member exists, and was specified correctly. Correct the specification or create the object and try the operation again.

AMQ8771 (DEC)
OpenVMS Cluster Failover Set Configuration and State.

Severity
0 : Information

AMQ8772 (DEC)
Queue Manager Name: <*insert_3*> Sequence No: <*insert_1*>

Severity
0 : Information

AMQ8773 (DEC)
TCP/IP Address: <*insert_3*> Listener Port Number : <*insert_4*>

Severity
0 : Information

AMQ8774 (DEC)
Queue Manager state in failover set: STARTED

Severity
0 : Information

AMQ8775 (DEC)
Queue Manager state in failover set: STOPPED

Severity
0 : Information

AMQ8776 (DEC)
Node specific configuration and state

Severity
0 : Information

AMQ8777 (DEC)
Node name: <*insert_3*> Priority: <*insert_1*> TCP/IP Interface: <*insert_4*>

Severity
0 : Information

AMQ8778 (DEC)
Queue Manager state : RUNNING

Severity
0 : Information

AMQ8779 (DEC)
Queue Manager state : AVAILABLE

Severity
0 : Information

AMQ8780 (DEC)
Queue Manager state : EXCLUDED

Severity
0 : Information

AMQ8781 (DEC)
Failover Monitor state: STARTED

Severity
0 : Information

AMQ8782 (DEC)
Failover Monitor state: STOPPED

Severity
0 : Information

AMQ8783 (DEC)
Failover Monitor state: WATCHING

Severity
0 : Information

AMQ8784 (DEC)
Node <*insert_3*> is not in the Failover Set configuration file

Severity
20 : Error

AMQ8785 (DEC)
There are no Failover Monitors started for Queue Manager: <*insert_3*>

Severity
20 : Error

AMQ8786 (DEC)
Failover set update operation in progress

Severity
10 : Warning

AMQ8787 (DEC)
Usage:

Start the queue manager in the failover set

failover -m <queue manager> [-n <node name>] -s

End the queue manager in the failover set

failover -m <queue manager> -e

Failover the running queue manager to another node

failover -m <queue manager> [-n <node name>] -f

Stop a failover monitor on a node

failover -m <queue manager> -n <node name> -h

Query the state of the queue manager

failover -m <queue manager> -q

Set the symbols MQS$QMGR_NODE, MQS$AVAILABLE_NODES and MQS$MONITOR_NODES

failover -m <queue manager> -l

Change the state of the failover set

failover -m <queue manager> -c -cluster stopped|started

Change the state of the queue manager on a node

failover -m <queue manager> -n <node name> -c -qmgr available|running|excluded

Change the state of the monitor on a node

failover -m <queue manager> -n <node name> -c -monitor stopped | started | watcher

Clear the update in progress flag

failover -m <queue manager> -u

Severity
0 : Information

AMQ8788 (DEC)
Usage: failover_monitor -m <queue manager> [-d]

Severity
0 : Information

AMQ8789 (DEC)
Error opening failover initialisation file FAILOVER.INI

Severity
20 : Error

AMQ8790 (DEC)
Error in the format of the initialisation file FAILOVER.INI

Severity
20 : Error

AMQ8791 (DEC)
No node available on which to start the queue manager

Severity
20 : Error

AMQ8792 (DEC)
Operation not allowed; Use a Failover command

Severity
20 : Error

AMQ8793 (DEC)
The ending of the queue manager was forced

Severity
10 : Warning

AMQ8794 (DEC)
The ending of the queue manager timed out before completion

Severity
20 : Error

AMQ8795 (DEC)
End Queue Manager Time Out: <*insert_1*>

Severity
0 : Information

AMQ8796 (DEC)
There is a Failover Monitor already running on node: <*insert_3*>

Severity
20 : Error

AMQ8797 (Tandem)
Cannot move queue files to <*insert_3*>.

Severity
0 : Information

Explanation
The MQSeries altmqfls utility cannot move the specified queue files to volume <*insert_3*>.

Response
Verify that the queue files are not already on volume <*insert_3*> using the dspmqfls utility. Verify that volume <*insert_3*> does not already contain queue files for this or any other queue manager in the same subvolume as used by this queue manager.

AMQ8798 (Tandem)
Queue files moved to <*insert_3*>.

Severity
0 : Information

Explanation
The MQSeries altmqfls utility has sucessfully moved the specified queue files to volume <*insert_3*>.

Response
None.

AMQ8801 (Tandem)
EC Boss <*insert_3*> for Queue Manager <*insert_4*> is initializing.

Severity
30 : Severe error

Explanation
The EC Boss for Queue Manager <*insert_4*> is beginning the startup sequence. The process name of the EC Boss is <*insert_3*>.

AMQ8802 (Tandem)
EC Boss <*insert_3*> for Queue Manager <*insert_4*> initialization complete.

Severity
30 : Severe error

Explanation
The EC Boss for Queue Manager <*insert_4*> has completed process startup actions. The process name of the EC Boss is <*insert_3*>.

AMQ8803 (Tandem)
EC Boss <*insert_3*> for Queue Manager <*insert_4*> controlled shutdown initiated.

Severity
30 : Severe error

Explanation
The EC Boss for Queue Manager <*insert_4*> has entered the controlled shutdown state. The Queue Manager will not accept new work, and once operations in progress have completed, connections will be terminated. When there are no more connections, the Queue Manager will end.

AMQ8804 (Tandem)
EC Boss <*insert_3*> for Queue Manager <*insert_4*> quiesce shutdown initiated.

Severity
30 : Severe error

Explanation

The EC Boss for Queue Manager <*insert_4*> has entered the quiesce shutdown state. The Queue Manager will not accept new work, but will allow existing connections to complete before ending.

AMQ8805 (Tandem)

EC Boss <*insert_3*> for Queue Manager <*insert_4*> immediate shutdown initiated.

Severity

30 : Severe error

Explanation

The EC Boss for Queue Manager <*insert_4*> has entered the immediate shutdown state. Any current connections are terminated and the Queue Manager will end immediately.

AMQ8806 (Tandem)

EC / EC Boss <*insert_3*> for Queue Manager <*insert_4*> cannot access file <*insert_5*>

Severity

40 : Stop Error

Explanation

An EC, or the EC Boss (process name <*insert_3*>) for Queue Manager <*insert_4*> has not been able to access the file named <*insert_5*>. This file is critical to the operation of the Queue Manager, and the Queue Manager will not start properly until the problem is corrected.

Response

End the Queue Manager and check the existence or file attributes of the file named <*insert_5*>. Verify that the file exists, and has the appropriate file security and type attributes, correct the problem and restart the Queue Manager.

AMQ8807 (Tandem)

EC / EC Boss <*insert_3*> for Queue Manager <*insert_4*> obtained file error <*insert_1*> on file <*insert_5*>

Severity

40 : Stop Error

Explanation

An EC, or the EC Boss (process name <*insert_3*>) for Queue Manager <*insert_4*> obtained Tandem file error <*insert_1*> while attempting an IO operation to file <*insert_5*>. The successful completion of the IO operation may be critical to the correct operation of the Queue Manager, and the Queue Manager may not operate properly until the problem is corrected.

Response

End the Queue Manager and check the file attributes of the file named <*insert_5*>. Verify that the file has the appropriate file security and type attributes, correct the problem and restart the Queue Manager.

AMQ8808 (Tandem)

Incorrect Queue Manager name <*insert_4*> supplied to process <*insert_4*>

Severity

40 : Stop Error

Explanation

A Queue Manager process (process name <*insert_3*>) was supplied with an invalid or non-existent Queue Manager name, <*insert_4*>. The initialization of the process failed as a result.

Response

End the Queue Manager and check the queue manager name that is being used in the configuration databases. After correcting the problem, restart the Queue Manager.

AMQ8809 (Tandem)
Queue Manager <*insert_4*> started.

Severity
30 : Severe error

Explanation
The EC Boss has reported that the Queue Manager named <*insert_4*> has entered the "started" state.

AMQ8810 (Tandem)
EC number <*insert_1*>, process name <*insert_3*>, for Queue Manager <*insert_4*> is initializing.

Severity
30 : Severe error

Explanation
An EC in the Queue Manager named <*insert_4*> has started and is performing process initialization.

AMQ8811 (Tandem)
EC number <*insert_1*>, process name <*insert_3*>, for Queue Manager <*insert_4*> has completed initialization.

Severity
30 : Severe error

Explanation
An EC in the Queue Manager named <*insert_4*> has completed process initialization.

AMQ8812 (Tandem)
EC number <*insert_1*>, process name <*insert_3*>, for Queue Manager <*insert_4*> has started controlled shutdown.

Severity
30 : Severe error

Explanation
An EC in the Queue Manager named <*insert_4*> has reported that a controlled shutdown has started. The EC will wait for all currently running agents to end before performing the final shutdown actions.

AMQ8813 (Tandem)
EC number <*insert_1*>, process name <*insert_3*>, for Queue Manager <*insert_4*> has started quiesce shutdown.

Severity
30 : Severe error

Explanation
An EC in the Queue Manager named <*insert_4*> has reported that a quiesce shutdown has started. The EC will wait for all currently running agents to end before performing the final shutdown actions.

AMQ8814 (Tandem)
EC number <*insert_1*>, process name <*insert_3*>, for Queue Manager <*insert_4*> has started immediate shutdown.

Severity
30 : Severe error

Explanation
An EC in the Queue Manager named <*insert_4*> has reported that an immediate shutdown has started. The EC will terminate immediately, without waiting for currently running agents to end.

AMQ8815 (Tandem)

EC number <*insert_1*>, process name <*insert_3*>, for Queue Manager <*insert_4*> has shutdown.

Severity

30 : Severe error

Explanation

An EC in the Queue Manager named <*insert_4*> has reported that it has completed shutdown actions. When all ECs in the Queue Manager have completed shutdown actions, the Queue Manager will end.

AMQ8816 (Tandem)

Queue Manager <*insert_4*> has started, though only <*insert_1*> of <*insert_2*> ECs have registered.

Severity

30 : Severe error

Explanation

The Queue Manager named <*insert_4*> has entered the started state, and will now accept connections. However, only <*insert_1*> of the expected <*insert_2*> ECs have registered with the EC Boss. The Queue manager's load balancing and overall performance will be adversly affected, however it will still be able to service connections.

Response

Examine the logs to determine the cause of the failure to start the missing ECs. End the Queue Manager, and rectify the problem if possible. Restart the Queue Manager and ensure that the Queue Manager starts correctly.

AMQ8817 (Tandem)

Process <*insert_3*> in Queue Manager <*insert_4*> cannot process a request due to a resource problem.

Severity

40 : Stop Error

Explanation

The process named <*insert_3*> has failed to process a request from another process due to a failure to allocate a resource, such as memory, or disk space. Depending upon the criticality of the resource itself, this may cause further errors, or the failure of certain Queue Manager components.

Response

Examine the logs to determine the cause of the failure. If there are resource problems that can be corrected, correct them and attempt the operation again.

AMQ8818 (Tandem)

EC Boss in Queue Manager <*insert_4*> rejected a registration from process <*insert_3*>.

Severity

40 : Stop Error

Explanation

The process named <*insert_3*> attempted to register with the EC Boss. The EC Boss detected a problem with the registration information and rejected the attempt.

Response

Examine the logs to determine further information about the problem. Determine the identity of the process, and verify that the process is an EC. If the process is not an EC, or cannot be identified, then a security threat may be present.

AMQ8819 (Tandem)

EC number <*insert_1*> registered with the EC Boss in Queue Manager <*insert_4*>.

Severity
40 : Stop Error

Explanation
EC number <*insert_1*> has registered with the EC Boss. When all the expected ECs in a Queue Manager have registered, the Queue Manager enters the started state.

AMQ8820 (Tandem)
An unknown message received by process <*insert_3*> in Queue Manager <*insert_4*> from process <*insert_5*> has been rejected.

Severity
40 : Stop Error

Explanation
The process <*insert_3*> has received and rejected a message that is either not of the correct format, or from an unknown source.

Response
Examine the log to see if further information is available. Try to identify the process to ensure that a security threat is not present.

AMQ8821 (Tandem)
The EC Boss in Queue Manager <*insert_4*> detected the failure of EC number <*insert_1*>.

Severity
40 : Stop Error

Explanation
The EC Boss has detected that EC number <*insert_1*> has terminated unexpectedly. If the maximum number of restarts performed on this EC has not already been exceeded, PATHWAY will attempt to restart the EC.

Response
Examine the log to see if further information is available.

AMQ8822
Invalid response, please re-enter (y or n):

Severity
0 : Information

Response
None.

AMQ8823 (Tandem)
Process <*insert_3*> in Queue Manager <*insert_4*> received and rejected a message from an unknown source, <*insert_5*>.

Severity
40 : Stop Error

Explanation
A process in Queue Manager <*insert_4*> received a message from a source that is not authorized or not registered to communicate with the Queue Manager. The process is identified by <*insert_5*>. The process that received the message is identified by <*insert_3*>.

Response
Examine the log to see if further information is available on the identity of the source of the message. Try to determine the identity of the sender and verify that no security threat is present.

AMQ8824 (Tandem)
The EC Boss in Queue Manager <*insert_4*> detected an inconsistency in the context data for agent process <*insert_3*>.

Severity
40 : Stop Error

Explanation
The EC Boss found that the information it had previously held about the agent <*insert_3*> is not consistent with new information.

Response
Examine the log to see if further information is available relating to process <*insert_3*>.

AMQ8825 (Tandem)
EC number <*insert_1*> in Queue Manager <*insert_4*> detected the failure of the EC Boss.

Severity
40 : Stop Error

Explanation
An EC detected that the EC Boss for the Queue Manager has failed. If the maximum number of restarts for the EC Boss has not been exceeded, PATHWAY will attempt to restart the EC Boss.

Response
Examine the log to see if further information is available relating to the failure of the EC Boss. If the problem persists, end the Queue Manager, correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8826 (Tandem)
EC number <*insert_1*> in Queue Manager <*insert_4*> detected the failure of an <*insert_5*> agent servicing <*insert_3*>.

Severity
40 : Stop Error

Explanation
An EC detected that an <*insert_5*> agent process for <*insert_3*> has failed. If the maximum number of restarts of agent processes has not already been exceeded, the EC will attempt to restart the agent process when it is required.

Response
Examine the log to see if further information is available relating to the failure of the agent process. If the problem persists, end the Queue Manager, correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8827 (Tandem)
EC number <*insert_1*> in Queue Manager <*insert_4*> failed to communicate with the EC Boss.

Severity
40 : Stop Error

Explanation
An EC attempted to communicate with the EC Boss, but the attempt failed. The failure to communicate is interpreted by the EC as EC Boss failure.

Response

Examine the log to see if further information is available relating to the failure to communicate with the EC Boss. If the problem persists, end the Queue Manager, correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8828 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to communicate with <*insert_5*> agent process <*insert_3*>.

Severity

40 : Stop Error

Explanation

An EC attempted to communicate with an agent process, but the attempt failed. The failure to communicate is interpreted by the EC as agent failure. Depending upon various factors, the EC may attempt to restart the agent.

Response

Examine the log to see if further information is available relating to the failure to communicate with the agent. If the problem persists, end the Queue Manager, correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8829 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to start an <*insert_5*> agent.

Severity

40 : Stop Error

Explanation

An EC attempted to create an agent process, but the attempt failed. If the maximum number of agent restarts has not already been exceeded, the EC will attempt to restart the agent process.

Response

Examine the log to see if further information is available relating to the failure to start the agent. If the problem persists, end the Queue Manager, correct the problem and restart. If the problem cannot be identified as a configuration problem, use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or thehttp://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ8830 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to service a Stop Channel request for channel <*insert_5*>.

Severity

40 : Stop Error

Explanation

An EC attempted to process a Stop Channel request, but the attempt failed. The failure will be relayed back to the original requestor via the EC Boss.

Response

Examine the log to see if further information is available relating to the failure to service the Stop Channel request. The originator of the Stop Channel request will be informed of the failure, togther with the reason for the failure.

AMQ8831 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to service an agent "done" request from agent process <*insert_3*>.

Severity

40 : Stop Error

Explanation

An EC attempted to process an agent "done" request, but the attempt failed. An agent "done" request indicates that agent process <*insert_3*> has completed its work and is asking the EC whether to terminate, or to go idle. For some reason, the EC failed to process the request. The EC will terminate the agent process.

Response

Examine the log to see if further information is available relating to the failure to service the agent "done" request.

AMQ8832 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> created an idle <*insert_5*> agent process <*insert_3*>.

Severity

30 : Severe error

Explanation

An EC successfully created an idle agent.

AMQ8833 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to activate <*insert_5*> agent process <*insert_3*>.

Severity

40 : Stop Error

Explanation

An EC failed to activate an idle agent in order to service a connection, or start channel request. The request could not be satisfied by the EC. The EC returns a failure completion and reason code to the originator of the request.

Response

Examine the log to see if further information is available relating to the failure to activate the agent.

AMQ8834 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to deactivate <*insert_5*> agent process <*insert_3*>.

Severity

40 : Stop Error

Explanation

An EC failed to deactivate an active agent after the agent indicated that it had completed processing a connection or channel.

Response

Examine the log to see if further information is available relating to the failure to deactivate the agent.

AMQ8835 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> destroyed idle <*insert_5*> agent process <*insert_3*>.

Severity

30 : Severe error

Explanation

An EC successfully destroyed an idle agent process. The EC normally performs this operation as a result of managing the pool of idle agents. Agents that have been used more than a certain (configurable) number of times are destroyed and a fresh agent created in their place.

AMQ8836 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to destroy an idle <*insert_5*> agent process <*insert_3*>.

Severity

40 : Stop Error

Explanation

An EC failed to destroy an idle agent process. The EC normally performs this operation as a result of managing the pool of idle agents. Agents that have been used more than a certain (configurable) number of times are destroyed and a fresh agent created in their place.

Response

Examine the log to see if further information is available relating to the failure to destroy the agent.

AMQ8837 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to create an idle <*insert_5*> agent.

Severity

40 : Stop Error

Explanation

An EC failed to create an idle <*insert_5*> agent process. The EC normally performs this operation as a result of managing the pool of idle agents. Agents that have been used more than a certain (configurable) number of times are destroyed and a fresh agent created in their place.

Response

Examine the log to see if further information is available relating to the failure to create the agent.

AMQ8838 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> initiated creation of an idle <*insert_5*> agent.

Severity

30 : Severe error

Explanation

An EC successfully initiated the creation of an idle <*insert_5*> agent process. The EC normally performs this operation as a result of managing the pool of idle agents. Agents that have been used more than a certain (configurable) number of times are destroyed and a fresh agent created in their place.

AMQ8839 (Tandem)

EC number <*insert_1*> in Queue Manager <*insert_4*> failed to complete a <*insert_3*> request for channel <*insert_5*>.

Severity

40 : Stop Error

Explanation
An EC failed to complete the processing of a <*insert_3*> request. The originator of the request is passed the completion status and reason code.

Response
Examine the log to see if further information is available relating to the failure to complete the processing of the request.

AMQ8840 (Tandem)
EC number <*insert_1*> in Queue Manager <*insert_4*> failed to complete an agent status request for agent process <*insert_3*>.

Severity
40 : Stop Error

Explanation
An EC failed to complete the processing of an agent status request. The EC Boss or EC has detected an inconsistency in context information about the agent.

Response
Examine the log to see if further information is available relating to the failure to complete the processing of the request.

AMQ8841 (Tandem)
EC process <*insert_3*> in Queue Manager <*insert_4*> is waiting for the EC Boss to initialize.

Severity
30 : Severe error

Explanation
An EC is waiting for the EC Boss to initialize and create its entry in the RUNTIME file for the Queue Manager.

AMQ8842 (Tandem)
Error attempting to create queue manager.

Severity
40 : Stop Error

Explanation
MQ verification request, omvStartChildProcess, failed.

Response
None.

AMQ8843 (Tandem)
Queue manager, <*insert_3*>, created successfully

Severity
0 : Information

Response
None.

AMQ8844 (Tandem)
Queue manager, <*insert_3*>, already created

Severity
0 : Information

Response
None.

AMQ8845 (Tandem)
An MQSeries NonStop Server has restarted its backup process

Severity
40 : Stop Error

Explanation
The MQSeries NonStop Server process <*insert_3*> detected the failure of its backup process and has restarted a new backup in CPU <*insert_1*>.

Response
Use the standard operating system facilities to diagnose the cause of the backup NonStop Server failure and attempt to correct it. MQSeries will continue without interruption.

AMQ8846 (Tandem)
MQSeries NonStop Server takeover initiated

Severity
40 : Stop Error

Explanation
The MQSeries NonStop Server backup process <*insert_3*> detected the failure of its primary process and is in the process of taking over and starting a new backup. The new NonStop Server primary process is now running in CPU <*insert_1*>.

Response
Use the standard operating system facilities to diagnose the cause of the primary NonStop Server failure and attempt to correct it. MQSeries will continue without interruption.

AMQ8847 (Tandem)
The EC Boss in Queue Manager <*insert_4*> failed to find an EC to service a request.

Severity
40 : Stop Error

Explanation
The EC Boss failed to find an active EC to service a request that was made, either by an application (in order to start a connection), or by an administration command (for example, to start or stop a channel). It is possible that all ECs in the Queue Manager have failed repeatedly, exceeding the maximum number of restarts allowed by PATHWAY.

Response
Examine the log to see if further information is available on the state of the Queue Manager. The Queue Manager will need to be ended and restarted.

AMQ8850 (Tandem)
Warning: MQSeries Licence Exception Detected MQSeries has detected that this environment exceeds the authorized licence registration. Please review your licence registration by running the installation program INSTMQM with the -l option and if necessary, obtain the required extra use-authorization from your program provider to avoid being in breach of your MQSeries licence agreement.

Severity
0 : Information

Explanation
None.

Response
None.

AMQ8851 (Tandem)
MQSeries CleanRDF utility has detected an error

Severity
40 : Stop Error

Explanation

CleanRDF (queue manager <*insert_5*>) encountered a(n) <*insert_4*> error on the rdfpurge file <*insert_3*>. The file system returned error code <*insert_1*>.

Response

Use the standard operating system facilities to verify the state of this file and reinvoke the utility if the error is deemed transient.

AMQ8852 (Tandem)

MQSeries CleanRDF utility has detected an error

Severity

40 : Stop Error

Explanation

CleanRDF (queue manager <*insert_5*>) has detected that the backup system <*insert_4*> is inaccessible. The file system returned error code <*insert_1*>.

Response

Contact your systems administrator and reinvoke the utility if the error is deemed transient.

AMQ8853 (Tandem)

MQSeries CleanRDF utility has detected an error

Severity

40 : Stop Error

Explanation

CleanRDF (queue manager <*insert_5*>) has encountered a TM/MP <*insert_4*> error. The system returned error code <*insert_1*>.

Response

Contact your systems administrator and reinvoke the utility if the error is deemed transient.

AMQ8854 (Tandem)

MQSeries CleanRDF utility has detected an error

Severity

40 : Stop Error

Explanation

CleanRDF (queue manager <*insert_5*>) encountered a(n) <*insert_4*> error on file <*insert_3*>. The system returned error code <*insert_1*>.

Response

Ensure that a file with this name exists on the same volume and subvolume (i.e. create if necessary - format is irrelevant) on both the primary system and backup systems before reinvoking the utility.

AMQ8855 (Tandem)

MQSeries CleanRDF utility has detected an error

Severity

40 : Stop Error

Explanation

CleanRDF (queue manager <*insert_5*>) encountered a(n) <*insert_4*> error for the FUP process <*insert_3*>. The system returned error code <*insert_1*>.

Response

Use the standard operating system facilities to verify the MQRDFFUPPROGNAME and MQRDFFUPPROCESSNAME environment parameters. Reinvoke the utility if the error is deemed transient.

AMQ8856 (Tandem)
MQSeries CleanRDF utility has detected an error

Severity
40 : Stop Error

Explanation
CleanRDF (queue manager <*insert_5*>) encountered an error when attempting to duplicate file <*insert_3*> to backup system <*insert_4*>. The system returned error code <*insert_1*>.

Response
Use the standard operating system facilities to verify the state of this file on both primary and backup systems. Reinvoke the utility if the error is deemed transient.

AMQ8857 (Tandem)
MQSeries CleanRDF utility STATISTICS Message

Severity
40 : Stop Error

Explanation
CleanRDF of queue manager <*insert_5*> has completed operation. <*insert_1*> files were deleted. <*insert_2*> files were Skipped. <*insert_3*> static files were duplicated to backup system <*insert_4*>.

AMQ8871
Entity, principal or group not known.

Severity
20 : Error

Explanation
The authorization entity, which can be either a principal or a group, could not be found.

AMQ8874 (Tandem)
Placeholder for new message

Severity
40 : Stop Error

Explanation
This is a placeholder for a new message

AMQ8875 (Tandem)
Placeholder for new message

Severity
40 : Stop Error

Explanation
This is a placeholder for a new message

AMQ8876 (Tandem)
Placeholder for new message

Severity
40 : Stop Error

Explanation
This is a placeholder for a new message

AMQ8877
WebSphere MQ channel authentication record set.

Severity
0 : Information

Explanation
WebSphere MQ channel authentication record set.

AMQ8878
Display channel authentication record details.

Severity
0 : Information

Explanation
The display channel authentication command completed successfully. Details follow this message.

AMQ8879
Channel authentication record type not valid.

Severity
20 : Error

Explanation
The type parameter specified on the command was not valid.

Response
Specify a valid type. Refer to the WebSphere MQ Script (MQSC) Command Reference to determine an allowable combination of parameters for this command.

AMQ8880
Channel authentication record action not valid.

Severity
20 : Error

Explanation
The action parameter specified on the command was not valid.

Response
Specify a valid action. Refer to the WebSphere MQ Script (MQSC) Command Reference to determine an allowable combination of parameters for this command.

AMQ8881
Channel authentication record user source not valid.

Severity
20 : Error

Explanation
The user source parameter specified on the command was not valid.

Response
Specify a valid user source. Refer to the WebSphere MQ Script (MQSC) Command Reference to determine an allowable combination of parameters for this command.

AMQ8882
Parameter not allowed for this channel authentication record type.

Severity
20 : Error

Explanation
The parameter is not allowed for the type of channel authentication record being set or displayed.

Response
Refer to the description of the parameter in error to determine the types of record for which this parameter is valid.

AMQ8883
Channel authentication record already exists.

Severity
20 : Error

Explanation
An attempt was made to add a channel authentication record, but it already exists.

Response
Specify action as MQACT_REPLACE.

AMQ8884
Channel authentication record not found.

Severity
20 : Error

Explanation
The specified channel authentication record does not exist.

Response
Specify a channel authentication record that exists.

AMQ8885
Parameter not allowed for this action on a channel authentication record.

Severity
20 : Error

Explanation
The parameter is not allowed for the action being applied to a channel authentication record. Refer to the description of the parameter in error to determine the actions for which this parameter is valid.

Response
Remove the parameter.

AMQ8886
Parameter not allowed for this channel authentication record user source value.

Severity
20 : Error

Explanation
The parameter is not allowed for a channel authentication record with the value that the user source field contains. Refer to the description of the parameter in error to determine the values of user source for which this parameter is valid.

Response
Remove the parameter.

AMQ8887
Parameter not allowed for this channel authentication record match value.

Severity
20 : Error

Explanation
The parameter is not allowed for an inquire channel authentication record with the value that the match field contains. Refer to the description of the parameter in error to determine the values of match for which this parameter is valid.

Response
Remove the parameter.

AMQ8888
Channel authentication record warn value not valid.

Severity
20 : Error

Explanation
The warn parameter specified on the command was not valid.

Response
Specify a valid value for warn. Refer to the WebSphere MQ Script (MQSC) Command Reference to determine an allowable combination of parameters for this command.

AMQ8891
Channel authentication profile name is invalid.

Severity
20 : Error

Explanation
The channel profile name used in the command was not valid. This may be because it contained characters which are not accepted in WebSphere MQ names, or characters which are not valid for the specified profile type.

Response
None.

AMQ8901 (Tandem)
A Status Server has started

Severity
0 : Information

Explanation
A Status Server in CPU <*insert_1*> has started. The process is named <*insert_3*>.

Response
None.

AMQ8902 (Tandem)
A Status Server has ended normally.

Severity
0 : Information

Explanation
A Status Server in CPU <*insert_1*> has ended normally. The process was named <*insert_3*>.

Response
None.

AMQ8903 (Tandem)
A Status Server has ended with errors.

Severity
0 : Information

Explanation
A Status Server in CPU <*insert_1*> has ended with errors. The process was named <*insert_3*>. The error return code reported by the Status Server is <*insert_2*>. The Status Server should be restarted automatically by the Queue Manager.

Response
Verify that the Status Server has restarted correctly. Examine the Queue Manager FD subvolume for FFST files that may have been generated by the Status Server. Use the process name to locate the relevant FFSTs. Attempt to reconstruct the chain of events or symptoms that lead to the failure and save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at

http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ8904 (Tandem)
A Status Server has detected a CPU failure.

Severity
0 : Information

Explanation
The Status Server process <*insert_3*> has detected that CPU <*insert_1*> failed. If there were components of the Status Manager that were running in this CPU, they will now no longer be available, and application connections and channels may be dropped. The Status Manager should continue to be available to new connections and channels. Any Status Server and Queue Server processes that were running in that CPU will be replaced in other available CPUs.

Response
None normally necessary. Applications could experience the reason code MQRC_CONNECTION_BROKEN (2009) from MQI operations in progress that used agent processes running in the failed CPUs, but they should be able to immediatelt re-connect successfully.

AMQ8905 (Tandem)
A Status Server completed takeover processing.

Severity
0 : Information

Explanation
The Status Server process <*insert_3*> has completed processing that was associated with a prior takeover from a failed primary Status Server process, or the failure of the CPU that it was running in. Normal processing resumes after this point, and the Status Server is again in a state where it is resilient to any single point of failure.

Response
None normally necessary. This message is logged to provide positive confirmation that the takeover is complete.

AMQ8906 (Tandem)
More Channel Status' hardened than Max allowed.

Severity
0 : Information

Explanation
There were more Channel Status' hardened to the STATABLE than the MAXACTIVECHANNELS in the QMINI File.

Response
None.

AMQ8919
There are no matching IBM WebSphere MQ queue manager names.

Severity
30 : Severe error

AMQ8934 (IBM i)
Message . . . . :

Severity
10 : Warning

AMQ8935 (IBM i)
Cause . . . . . :

Severity
10 : Warning

AMQ8936 (IBM i)
Recovery . . . :

Severity
10 : Warning

AMQ8937 (IBM i)
Technical Description . . . . . :

Severity
10 : Warning

AMQ8A01 (IBM i)
Create Message Queue Manager

AMQ8A02 (IBM i)
Delete Message Queue Manager

AMQ8A04 (IBM i)
Work with MQ Messages

AMQ8A05 (IBM i)
Change Message Queue Manager

AMQ8A06 (IBM i)
Display Message Queue Manager

AMQ8A07 (IBM i)
End Message Queue Manager

AMQ8A08 (IBM i)
Start Message Queue Manager

AMQ8A09 (IBM i)
Change MQ Queue

AMQ8A0A (IBM i)
Clear MQ Queue

AMQ8A0B (IBM i)
Copy MQ Queue

AMQ8A0C (IBM i)
Create MQ Queue

AMQ8A0D (IBM i)
Delete MQ Queue

AMQ8A0E (IBM i)
Display MQ Queue

AMQ8A0F (IBM i)
Work with MQ Queues

AMQ8A10 (IBM i)
Change MQ Process

AMQ8A11 (IBM i)
Copy MQ Process

AMQ8A12 (IBM i)
Create MQ Process

AMQ8A13 (IBM i)
Delete MQ Process

AMQ8A14 (IBM i)
Display MQ Process

AMQ8A15 (IBM i)
Work with MQ Processes

AMQ8A16 (IBM i)
Start MQ Command Server

AMQ8A17 (IBM i)
End MQ Command Server

AMQ8A18 (IBM i)
Display MQ Command Server

AMQ8A19 (IBM i)
Set MQ

AMQ8A20 (IBM i)
Quiesce Message Queue Managers

AMQ8A21 (IBM i)
Quiesce Retry Delay

AMQ8A23 (IBM i)
Work with Queue Status

AMQ8A30 (IBM i)
Create MQ Channel

AMQ8A31 (IBM i)
Display MQ Channel

AMQ8A32 (IBM i)
Start MQ Listener

AMQ8A33 (IBM i)
Ping MQ Channel

AMQ8A34 (IBM i)
Delete MQ Channel

AMQ8A36 (IBM i)
Work with MQ Channels

AMQ8A37 (IBM i)
Change MQ Channel

AMQ8A38 (IBM i)
Copy MQ Channel

AMQ8A39 (IBM i)
Reset MQ Channel

AMQ8A40 (IBM i)
End MQ Channel

AMQ8A41 (IBM i)
Start MQ Channel

AMQ8A42 (IBM i)
Start MQ Channel Initiator

AMQ8A43 (IBM i)
Grant MQ Object Authority

AMQ8A44 (IBM i)
Revoke MQ Object Authority

AMQ8A45 (IBM i)
Display MQ Object Authority

AMQ8A46 (IBM i)
Display MQ Object Names

AMQ8A47 (IBM i)
Refresh IBM WebSphere MQ Authority

AMQ8A48 (IBM i)
Work with MQ Authority

AMQ8A49 (IBM i)
Start MQ Service

AMQ8A50 (IBM i)
End MQ Service

AMQ8A51 (IBM i)
Connect MQ

AMQ8A52 (IBM i)
Disconnect MQ

AMQ8A53 (IBM i)
Work with MQ Authority Data

AMQ8A54 (IBM i)
Resolve MQ Channel

AMQ8A55 (IBM i)
Work with MQ Channel Status

AMQ8A56 (IBM i)
SSL Client Authentication

AMQ8A57 (IBM i)
SSL CipherSpec

AMQ8A58 (IBM i)
SSL Peer name

AMQ8A59 (IBM i)
Local communication address

AMQ8A5A (IBM i)
Batch Heartbeat Interval

AMQ8A5B (IBM i)
Remove Queues

AMQ8A5C (IBM i)
Refresh Repository

AMQ8A5D (IBM i)
IP Address

AMQ8A60 (IBM i)
Cluster Name

AMQ8A61 (IBM i)
Cluster Name List

AMQ8A62 (IBM i)
Mode Name

AMQ8A63 (IBM i)
Password

AMQ8A64 (IBM i)
Transaction Program Name

AMQ8A65 (IBM i)
User Profile

AMQ8A66 (IBM i)
Network Connection Priority

AMQ8A67 (IBM i)
Batch Interval

AMQ8A68 (IBM i)
Batch Interval

AMQ8A69 (IBM i)
Cluster Workload Exit Data

AMQ8A6A (IBM i)
Cluster Workload Exit

AMQ8A6B (IBM i)
Repository Cluster

AMQ8A6C (IBM i)
Repository Cluster Namelist

AMQ8A6D (IBM i)
Cluster Workload Exit Data Length

AMQ8A6E (IBM i)
Maximum Message Length

AMQ8A6F (IBM i)
Default Queue Manager

AMQ8A70 (IBM i)
Default Binding

AMQ8A71 (IBM i)
Channel Table

AMQ8A72 (IBM i)
Change MQ Namelist

AMQ8A73 (IBM i)
List of Names

AMQ8A74 (IBM i)
Namelist

AMQ8A75 (IBM i)
Create MQ Namelist

AMQ8A76 (IBM i)
recreate MQ Object

AMQ8A77 (IBM i)
Record MQ Object Image

AMQ8A78 (IBM i)
Start IBM WebSphere MQ Commands

AMQ8A7A (IBM i)
Copy MQ Namelist

AMQ8A7B (IBM i)
From Namelist

AMQ8A7C (IBM i)
To Namelist

AMQ8A7D (IBM i)
Delete MQ Namelist

AMQ8A7E (IBM i)
Display MQ Namelist

AMQ8A7F (IBM i)
Work with MQ Namelist

AMQ8A80 (IBM i)
Group Profile

AMQ8A81 (IBM i)
User Profile

AMQ8A82 (IBM i)
Service Component

AMQ8A83 (IBM i)
Work with MQ Queue Manager

AMQ8A84 (IBM i)
Work with MQ Clusters

AMQ8A85 (IBM i)
Start MQ Trigger Monitor

AMQ8A86 (IBM i)
End MQ Listeners

AMQ8A87 (IBM i)
Work with MQ Transactions

AMQ8A88 (IBM i)
Resolve MQ Transaction

AMQ8A89 (IBM i)
Work with MQ Cluster Queues

AMQ8A8A (IBM i)
Display Journal Receiver Data

AMQ8A8B (IBM i)
Start MQ Pub/Sub Broker

AMQ8A8C (IBM i)
End MQ Pub/Sub Broker

AMQ8A8D (IBM i)
Display MQ Pub/Sub Broker

AMQ8A8E (IBM i)
Clear MQ Pub/Sub Broker

AMQ8A8F (IBM i)
Delete MQ Pub/Sub Broker

AMQ8B01 (IBM i)
Message Queue Manager name

AMQ8B02 (IBM i)
Text 'description'

AMQ8B03 (IBM i)
Trigger interval

AMQ8B04 (IBM i)
Undelivered message queue

AMQ8B05 (IBM i)
Default transmission queue

AMQ8B06 (IBM i)
Maximum handle limit

AMQ8B07 (IBM i)
Maximum uncommitted messages

AMQ8B08 (IBM i)
Queue name

AMQ8B09 (IBM i)
Output

AMQ8B0A (IBM i)
Library

AMQ8B0B (IBM i)
File to receive output

AMQ8B0C (IBM i)
OPTION(*MVS) not valid without specifying a value for WAIT.

Severity
40 : Stop Error

Explanation
The OPTION(*MVS) parameter may not be specified without specifying a value for the WAIT parameter.

Response
Remove the OPTION(*MVS) parameter from the command or, specify a value for the WAIT parameter. Then try the command again.

AMQ8B0D (IBM i)
Member to receive output

AMQ8B0E (IBM i)
Replace or add records

AMQ8B0F (IBM i)
Option

AMQ8B10 (IBM i)
Mode

AMQ8B11 (IBM i)
Put enabled

AMQ8B12 (IBM i)
Default message priority

AMQ8B13 (IBM i)
Default message persistence

AMQ8B14 (IBM i)
Process name

AMQ8B15 (IBM i)
Triggering enabled

AMQ8B16 (IBM i)
Get enabled

AMQ8B17 (IBM i)
Sharing enabled

AMQ8B18 (IBM i)
Default share option

AMQ8B19 (IBM i)
Message delivery sequence

AMQ8B1A (IBM i)
Harden backout count

AMQ8B1B (IBM i)
Trigger type

AMQ8B1C (IBM i)
Trigger depth

AMQ8B1D (IBM i)
Trigger message priority

AMQ8B1E (IBM i)
Trigger data

AMQ8B1F (IBM i)
Retention interval

AMQ8B20 (IBM i)
Maximum queue depth

AMQ8B21 (IBM i)
Maximum message length

AMQ8B22 (IBM i)
Backout threshold

AMQ8B23 (IBM i)
Backout requeue name

AMQ8B24 (IBM i)
Initiation queue

AMQ8B25 (IBM i)
Usage

AMQ8B26 (IBM i)
Definition type

AMQ8B27 (IBM i)
Target object

AMQ8B28 (IBM i)
Remote queue

AMQ8B29 (IBM i)
Remote Message Queue Manager

AMQ8B2A (IBM i)
Transmission queue

AMQ8B2B (IBM i)
From queue name

AMQ8B2C (IBM i)
To queue name

AMQ8B2D (IBM i)
Replace

AMQ8B2E (IBM i)
Queue type

AMQ8B2F (IBM i)
Application type

AMQ8B30 (IBM i)
Application identifier

AMQ8B31 (IBM i)
User data

AMQ8B32 (IBM i)
Environment data

AMQ8B33 (IBM i)
From process

AMQ8B34 (IBM i)
To process

AMQ8B36 (IBM i)
Job name

AMQ8B37 (IBM i)
Number

AMQ8B3A (IBM i)
Convert message

AMQ8B3B (IBM i)
Replace to member

AMQ8B3C (IBM i)
Heartbeat interval

AMQ8B3D (IBM i)
Non Persistent Message Speed

AMQ8B3E (IBM i)
Force

AMQ8B3F (IBM i)
No Jobs to display

AMQ8B41 (IBM i)
Queue definition scope

AMQ8B42 (IBM i)
Queue depth high threshold

AMQ8B43 (IBM i)
Queue depth low threshold

AMQ8B44 (IBM i)
Queue full events enabled

AMQ8B45 (IBM i)
Queue high events enabled

AMQ8B46 (IBM i)
Queue low events enabled

AMQ8B47 (IBM i)
Service interval

AMQ8B48 (IBM i)
Service interval events

AMQ8B49 (IBM i)
Distribution list support

AMQ8B4A (IBM i)
Parent Message Queue Manager

AMQ8B4B (IBM i)
Break Parent link

AMQ8B4C (IBM i)
Child Message Queue Manager

AMQ8B53 (IBM i)
Authorization events enabled

AMQ8B54 (IBM i)
Inhibit events enabled

AMQ8B55 (IBM i)
Local error events enabled

AMQ8B56 (IBM i)
Remote error events enabled

AMQ8B57 (IBM i)
Performance events enabled

AMQ8B58 (IBM i)
Start and stop events enabled

AMQ8B59 (IBM i)
Automatic Channel Definition

AMQ8B5A (IBM i)
Auto Chan. Def. events enabled

AMQ8B5B (IBM i)
Auto Chan. Def. exit program

AMQ8B5C (IBM i)
Redefine system objects

AMQ8B5D (IBM i)
Wait time

AMQ8B5E (IBM i)
Startup Status Detail

AMQ8B60 (IBM i)
Transaction type

AMQ8B61 (IBM i)
Log recovery events enabled

AMQ8B62 (IBM i)
IP protocol

AMQ8B63 (IBM i)
Configuration events enabled

AMQ8B64 (IBM i)
Refresh Message Queue Manager

AMQ8B65 (IBM i)
Refresh Type

AMQ8B66 (IBM i)
Include Interval

AMQ8B67 (IBM i)
IBM WebSphere MQ queue manager refreshed.

AMQ8B68 (IBM i)
Channel events enabled

AMQ8B69 (IBM i)
SSL events enabled

AMQ8B6A (IBM i)
Filter command

AMQ8B6B (IBM i)
Filter keyword

AMQ8B6C (IBM i)
Filter operator

AMQ8B6D (IBM i)
Filter value

AMQ8B6E (IBM i)
Filter value <*insert_3*> not valid with keyword <*insert_4*>.

Severity
30 : Severe error

Explanation
The filter value <*insert_3*> is not valid with the keyword <*insert_4*>.

Response
Specify a valid filter value for the keyword <*insert_4*>.

AMQ8B70 (IBM i)
Change MQ AuthInfo object

AMQ8B71 (IBM i)
Copy MQ AuthInfo object

AMQ8B72 (IBM i)
Create MQ AuthInfo object

AMQ8B73 (IBM i)
Delete MQ AuthInfo object

AMQ8B74 (IBM i)
Display MQ AuthInfo object

AMQ8B75 (IBM i)
From AuthInfo name

AMQ8B76 (IBM i)
AuthInfo name

AMQ8B77 (IBM i)
AuthInfo type

AMQ8B78 (IBM i)
User name

AMQ8B79 (IBM i)
User password

AMQ8B7A (IBM i)
Work with AuthInfo objects

AMQ8B7B (IBM i)
To AuthInfo name

AMQ8B80 (IBM i)
Change MQ Processor Allowance

AMQ8B81 (IBM i)
Display MQ Processor Allowance

AMQ8B82 (IBM i)
Sufficient Licence Units

AMQ8C01 (IBM i)
From channel

AMQ8C02 (IBM i)
Channel name

AMQ8C03 (IBM i)
Channel type

AMQ8C04 (IBM i)
SSL key reset count

AMQ8C05 (IBM i)
Remote queue manager

AMQ8C07 (IBM i)
Transmission queue

AMQ8C08 (IBM i)
Connection name

AMQ8C09 (IBM i)
Message channel agent

AMQ8C10 (IBM i)
Message channel agent user ID

AMQ8C12 (IBM i)
Batch size

AMQ8C13 (IBM i)
Disconnect interval

AMQ8C14 (IBM i)
Short retry count

AMQ8C15 (IBM i)
Short retry interval

AMQ8C16 (IBM i)
Long retry count

AMQ8C17 (IBM i)
Long retry interval

AMQ8C18 (IBM i)
Security exit

AMQ8C19 (IBM i)
Message exit

AMQ8C20 (IBM i)
Send exit

AMQ8C21 (IBM i)
Receive exit

AMQ8C22 (IBM i)
SSL CRL Namelist

AMQ8C23 (IBM i)
SSL Key Repository

AMQ8C24 (IBM i)
Put authority

AMQ8C25 (IBM i)
Sequence number wrap

AMQ8C27 (IBM i)
Transport type

AMQ8C28 (IBM i)
Data count

AMQ8C29 (IBM i)
Count

AMQ8C30 (IBM i)
To channel

AMQ8C31 (IBM i)
Message sequence number

AMQ8C32 (IBM i)
SSL Cryptographic Hardware

AMQ8C33 (IBM i)
Security exit user data

AMQ8C34 (IBM i)
Send exit user data

AMQ8C35 (IBM i)
Receive exit user data

AMQ8C36 (IBM i)
Message exit user data

AMQ8C37 (IBM i)
Resolve option

AMQ8C38 (IBM i)
Connection name

AMQ8C39 (IBM i)
Transmission queue name

AMQ8C40 (IBM i)
SSL Repository Password

AMQ8C41 (IBM i)
First Message

AMQ8C42 (IBM i)
Maximum number of messages

AMQ8C43 (IBM i)
Maximum message size

AMQ8C44 (IBM i)
Message retry exit

AMQ8C45 (IBM i)
Message retry exit data

AMQ8C46 (IBM i)
Number of message retries

AMQ8C47 (IBM i)
Message retry interval

AMQ8C48 (IBM i)
Coded Character Set

AMQ8C49 (IBM i)
Max message length

AMQ8C50 (IBM i)
Repository name

AMQ8C51 (IBM i)
Repository name list

AMQ8C52 (IBM i)
Cluster workload exit length

AMQ8C53 (IBM i)
Cluster workload exit

AMQ8C54 (IBM i)
Cluster workload exit data

AMQ8C55 (IBM i)
Suspend Cluster Queue Manager

AMQ8C56 (IBM i)
Reset Cluster

AMQ8C57 (IBM i)
Refresh MQ Cluster

AMQ8C58 (IBM i)
Resume Cluster Queue Manager

AMQ8C59 (IBM i)
Action

AMQ8C5A (IBM i)
Queue Manager Name for removal

AMQ8C5B (IBM i)
Work with MQ Listeners

AMQ8C5C (IBM i)
Queue Manager Id for removal

AMQ8C60 (IBM i)
Display Cluster Message Queue Manager

AMQ8C61 (IBM i)
Cluster Queue Manager name

AMQ8C62 (IBM i)
End MQ Listeners

AMQ8C63 (IBM i)
Port number

AMQ8C64 (IBM i)
Message channel agent Type

AMQ8C65 (IBM i)
Task user identifier

AMQ8D01 (IBM i)
Trace MQ

AMQ8D02 (IBM i)
Trace option setting

AMQ8D03 (IBM i)
Trace level

AMQ8D04 (IBM i)
Trace types

AMQ8D05 (IBM i)
Maximum storage to use

AMQ8D06 (IBM i)
Trace early

AMQ8D07 (IBM i)
Exclude types

AMQ8D08 (IBM i)
Trace interval

AMQ8D0A (IBM i)
Output member options

AMQ8D10 (IBM i)
Object name

AMQ8D11 (IBM i)
Object type

AMQ8D12 (IBM i)
User names

AMQ8D13 (IBM i)
Authority

AMQ8D14 (IBM i)
Authorization list

AMQ8D15 (IBM i)
Reference object name

AMQ8D16 (IBM i)
Reference object type

AMQ8D17 (IBM i)
Object name

AMQ8D18 (IBM i)
Process name

AMQ8D19 (IBM i)
Queue name

AMQ8D1A (IBM i)
Queue Manager Library

AMQ8D1B (IBM i)
ASP Number

AMQ8D1C (IBM i)
Journal receiver threshold

AMQ8D1D (IBM i)
Journal buffer size

AMQ8D20 (IBM i)
Channel name

AMQ8D22 (IBM i)
Cluster name

AMQ8D23 (IBM i)
Cluster namelist name

AMQ8D24 (IBM i)
User name

AMQ8D25 (IBM i)
Channel status

AMQ8D26 (IBM i)
End connected jobs

AMQ8D27 (IBM i)
Timeout interval (seconds)

AMQ8D28 (IBM i)
Object/Profile name

AMQ8D29 (IBM i)
Service Component name

AMQ8D2A (IBM i)
Work with MQ Topics

AMQ8D2B (IBM i)
Topic name

AMQ8D2C (IBM i)
No topics to display

AMQ8D2D (IBM i)
Delete MQ Topic

AMQ8D2E (IBM i)
Display MQ Topic

AMQ8D30 (IBM i)
Keep Alive Interval

AMQ9000-9999: Remote

AMQ9001
Channel <*insert_3*> ended normally.

Severity
0 : Information

Explanation
Channel <*insert_3*> ended normally.

Response
None.

AMQ9002
Channel <*insert_3*> is starting.

Severity
0 : Information

Explanation
Channel <*insert_3*> is starting.

Response
None.

AMQ9003 (IBM i)
Channel <*insert_3*> last message sequence number is <*insert_1*>.

Severity
0 : Information

Explanation
Channel <*insert_3*> last message sequence number is <*insert_1*>.

Response
None.

AMQ9004 (IBM i)
Channel <*insert_3*> status information.

Severity
0 : Information

Explanation
Channel <*insert_3*> status information: Number of Messages in Doubt - <*insert_1*> In Doubt Sequence Number - <*insert_2*> In Doubt Logic Unit of Work ID - <*insert_4*>

Response
None.

AMQ9181
The response set by the exit is not valid.

Severity
30 : Severe error

Explanation
The user exit <*insert_3*> returned a response code <*insert_1*> that is not valid in the ExitResponse field of the channel exit parameters (MQCXP). Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program set a response code that is not valid.

AMQ9182
The secondary response set by the exit is not valid.

Severity
30 : Severe error

Explanation
The user exit <*insert_3*> returned a secondary response code <*insert_1*> in the ExitResponse2 field of the channel exit parameters (MQCXP) that is not valid. Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program set a secondary response code that is not valid.

AMQ9184
The exit buffer address set by the exit is not valid.

Severity
30 : Severe error

Explanation
The user exit <*insert_3*> returned an address <*insert_1*> for the exit buffer that is not valid, when the secondary response code in the ExitResponse2 field of the channel exit parameters (MQCXP) is set to MQXR2_USE_EXIT_BUFFER. Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program set an exit buffer address that is not valid. The most likely cause is the failure to set a value, so that the value is 0.

AMQ9185
The exit space set by the exit is not valid.

Severity
30 : Severe error

Explanation
The user exit <*insert_3*> returned an exit space value <*insert_1*> that is not valid in the ExitSpace field of the channel exit parameters (MQCXP). Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program set an exit space value that is not valid. Correct the error.

AMQ9186
Too much exit space reserved by send exits.

Severity
30 : Severe error

Explanation
At exit initialization the send exits in the send exit chain for channel <*insert_3*> returned values in the ExitSpace field of the channel exit parameters (MQCXP). The total of these ExitSpace values is <*insert_1*>. The maximum number of bytes that can be sent in a single transmission is <*insert_2*>. Room must be left for at least 1024 bytes of message data in each transmission. So too much exit space has been reserved by the send exits. The channel stops.

Response
Investigate why the send exit programs set exit space values that are too large. Correct the error.

AMQ9187
The header compression value set by the exit is not valid.

Severity
30 : Severe error

Explanation
The user exit <*insert_3*> returned a header compression value <*insert_1*> in the CurHdrCompression field of the channel exit parameters (MQCXP) that was not one of the negotiated supported values specified in the HdrCompList field of the channel description (MQCD). Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program specified a header compression value that was not one of the negotiated supported values.

AMQ9188
The message compression value set by the exit is not valid.

Severity
30 : Severe error

Explanation
The user exit <*insert_3*> returned a message compression value <*insert_1*> in the CurMsgCompression field of the channel exit parameters (MQCXP) that was not one of the negotiated supported values specified in the MsgCompList field of the channel description (MQCD). Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program specified a message compression value that was not one of the negotiated supported values.

AMQ9189
The data length set by the exit is not valid.

Severity
30 : Severe error

Explanation
The user exit <*insert_3*> returned a data length value <*insert_1*> that was not greater than zero. Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program set a data length that is not valid.

AMQ9190
Channel stopping because of an error in the exit.

Severity
30 : Severe error

Explanation
The user exit <*insert_3*>, invoked for channel <*insert_4*> with id <*insert_1*> and reason <*insert_2*>, returned values that are not valid, as reported in the preceding messages. The channel stops.

Response
Investigate why the user exit program set values that are not valid.

AMQ9195
Data length larger than maximum segment length.

Severity
30 : Severe error

Explanation
The data length <*insert_1*> set by send exit <*insert_3*> is larger than the maximum segment length (<*insert_2*>). The maximum segment length is the maximum number of bytes that can be sent in a single transmission minus the user exit space required by all the send exits subsequent to the current one in the send exit chain. Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program set a data length that is not valid. Correct the error.

AMQ9196
Data length is larger than the agent buffer length.

Severity
30 : Severe error

Explanation
The data length <*insert_1*> set by exit <*insert_3*> is larger than the agent buffer length. The user exit returned data in the supplied agent buffer, but the length specified is greater than the length of the buffer. Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program set a data length that is not valid. Correct the error.

AMQ9197
Data length is larger than the exit buffer length.

Severity
30 : Severe error

Explanation
The data length <*insert_1*> set by exit <*insert_3*> is larger than the exit buffer length. The user exit returned data in the supplied exit buffer, but the length specified is greater than the length of the buffer. Message AMQ9190 is issued giving more details, and the channel stops.

Response
Investigate why the user exit program set a data length that is not valid.

AMQ9201
Allocate failed to host <*insert_3*>.

Severity
30 : Severe error

Explanation
The attempt to allocate a conversation using <*insert_4*> to host <*insert_3*> was not successful.

Response
The error may be due to an incorrect entry in the <*insert_4*> parameters contained in the channel

definition to host <*insert_3*>. Correct the error and try again. If the error persists, record the error values and contact your systems administrator. The return code from the <*insert_4*> <*insert_5*> call was <*insert_1*> (X<*insert_2*>). It may be possible that the listening program at host <*insert_3*> is not running. If this is the case, perform the relevant operations to start the listening program for protocol <*insert_4*> and try again.

AMQ9202

Remote host <*insert_3*> not available, retry later.

Severity

30 : Severe error

Explanation

The attempt to allocate a conversation using <*insert_4*> to host <*insert_3*> was not successful. However the error may be a transitory one and it may be possible to successfully allocate a <*insert_4*> conversation later.

Response

Try the connection again later. If the failure persists, record the error values and contact your systems administrator. The return code from <*insert_4*> is <*insert_1*> (X<*insert_2*>). The reason for the failure may be that this host cannot reach the destination host. It may also be possible that the listening program at host <*insert_3*> was not running. If this is the case, perform the relevant operations to start the <*insert_4*> listening program, and try again.

AMQ9203

A configuration error for <*insert_4*> occurred.

Severity

30 : Severe error

Explanation

Error in configuration for communications to host <*insert_3*> . Allocation of a <*insert_4*> conversation to host <*insert_3*> was not possible.

Response

The configuration error may be one of the following:

1.If the communications protocol is LU 6.2, it may be that one of the transmission parameters (Mode, or TP Name) is incorrect. Correct the error and try again. The mode name should be the same as the mode defined on host <*insert_3*>. The TP name on <*insert_3*> should be defined.

2.If the communications protocol is LU 6.2, it may be that an LU 6.2 session has not been established. Contact your systems administrator.

3.If the communications protocol is TCP/IP, it may be that the host name specified is incorrect. Correct the error and try again.

4.If the communications protocol is TCP/IP, it may be that the host name specified cannot be resolved to a network address. The host name may not be in the name server.

The return code from the <*insert_4*><*insert_5*> call was <*insert_1*> (X<*insert_2*>).

Record the error values and tell the system administrator.

AMQ9204

Connection to host <*insert_3*> rejected.

Severity

30 : Severe error

Explanation

Connection to host <*insert_3*> over <*insert_4*> was rejected.

Response

The remote system may not be configured to allow connections from this host. Check the <*insert_4*> listener program has been started on host <*insert_3*>.

If the conversation uses LU 6.2, it is possible that either the User ID or Password supplied to the remote host is incorrect.

If the conversation uses TCP/IP, it is possible that the remote host does not recognize the local host as a valid host.

The return code from the <*insert_4*><*insert_5*> call was <*insert_1*> X(<*insert_2*>).

Record the error values and tell the systems administrator.

AMQ9205

The host name supplied is not valid.

Severity

30 : Severe error

Explanation

The supplied <*insert_4*> host name <*insert_3*> could not be resolved into a network address. Either the name server does not contain the host, or the name server was not available.

Response

Check the <*insert_4*> configuration on your host.

AMQ9206

Error sending data to host <*insert_3*>.

Severity

30 : Severe error

Explanation

An error occurred sending data over <*insert_4*> to <*insert_3*>. This may be due to a communications failure.

Response

The return code from the <*insert_4*> <*insert_5*> call was <*insert_1*> X(<*insert_2*>). Record these values and tell your systems administrator.

AMQ9207

The data received from host <*insert_3*> is not valid.

Severity

30 : Severe error

Explanation

Incorrect data format received from host <*insert_3*> over <*insert_4*>. It may be that an unknown host is attempting to send data. An FFST file has been generated containing the invalid data received.

Response

Tell the systems administrator.

AMQ9208

Error on receive from host <*insert_3*>.

Severity

30 : Severe error

Explanation

An error occurred receiving data from <*insert_3*> over <*insert_4*>. This may be due to a communications failure.

Response
The return code from the <*insert_4*> <*insert_5*> call was <*insert_1*> (X<*insert_2*>). Record these values and tell the systems administrator.

AMQ9209
Connection to host <*insert_3*> closed.

Severity
30 : Severe error

Explanation
An error occurred receiving data from <*insert_3*> over <*insert_4*>. The connection to the remote host has unexpectedly terminated.

Response
Tell the systems administrator.

AMQ9210
Remote attachment failed.

Severity
30 : Severe error

Explanation
There was an incoming attachment from a remote host, but the local host could not complete the bind.

Response
The return code from the <*insert_4*> <*insert_5*> call was <*insert_1*> (X<*insert_2*>). Record these values and tell the systems administrator who should check the <*insert_4*> configuration.

AMQ9211
Error allocating storage.

Severity
30 : Severe error

Explanation
The program was unable to obtain enough storage.

Response
Stop some programs which are using storage and retry the operation. If the problem persists contact your systems administrator.

AMQ9212
A TCP/IP socket could not be allocated.

Severity
30 : Severe error

Explanation
A TCP/IP socket could not be created, possibly because of a storage problem.

Response
The return code from the <*insert_4*> <*insert_5*> call was <*insert_1*> (X<*insert_2*>). Try the program again. If the failure persists, record the error values and tell the systems administrator.

AMQ9213
A communications error for <*insert_4*> occurred.

Severity
30 : Severe error

Explanation
An unexpected error occurred in communications.

Response
The return code from the <*insert_4*> <*insert_5*> call was <*insert_1*> (X<*insert_2*>). Record these values and tell the systems administrator.

AMQ9214
Attempt to use an unsupported communications protocol.

Severity
30 : Severe error

Explanation
An attempt was made to use an unsupported communications protocol type <*insert_2*>.

Response
Check the channel definition file. It may be that the communications protocol entered is not a currently supported one.

AMQ9215
Communications subsystem unavailable.

Severity
30 : Severe error

Explanation
An attempt was made to use the communications subsystem, but it has not been started.

Response
Start the communications subsystem, and rerun the program.

AMQ9216
Usage: <*insert_3*> [-m QMgrName] [-n TPName]

Severity
20 : Error

Explanation
Values passed to the responder channel program are not valid. The parameters that are not valid are as follows :-

<*insert_4*>

The responder channel program exits.

Response
Correct the parameters passed to the channel program and retry the operation.

AMQ9216 (AIX)
Usage: <*insert_3*> [-m QMgrName]

Severity
20 : Error

Explanation
Values passed to the responder channel program are not valid. The parameters that are not valid are as follows :-

<*insert_4*>

The responder channel program exits.

Response
Correct the parameters passed to the channel program and retry the operation.

AMQ9216 (HP-UX)
Usage: <*insert_3*> [-m QMgrName]

Severity
20 : Error

Explanation
Values passed to the responder channel program are not valid. The parameters that are not valid are as follows :-

<insert_4>

The responder channel program exits.

Response
Correct the parameters passed to the channel program and retry the operation.

AMQ9217
The TCP/IP listener program could not be started.

Severity
30 : Severe error

Explanation
An attempt was made to start a new instance of the listener program, but the program was rejected.

Response
The failure could be because either the subsystem has not been started (in this case you should start the subsystem), or there are too many programs waiting (in this case you should try to start the listener program later).

AMQ9218
The *<insert_4>* listener program could not bind to port number *<insert_1>*.

Severity
30 : Severe error

Explanation
An attempt to bind the *<insert_4>* socket to the listener port was unsuccessful.

Response
The failure could be due to another program using the same port number. The return code from the *<insert_3>* call for port *<insert_5><insert_1>* was *<insert_2>*. Record these values and tell the systems administrator.

AMQ9219
The TCP/IP listener program could not create a new connection for the incoming conversation.

Severity
30 : Severe error

Explanation
An attempt was made to create a new socket because an attach request was received, but an error occurred.

Response
The failure may be transitory, try again later. If the problem persists, record the return code *<insert_1>* and tell the systems administrator. It may be necessary to free some jobs, or restart the communications system.

AMQ9220
The *<insert_4>* communications program could not be loaded.

Severity
30 : Severe error

Explanation
The attempt to load the <*insert_4*> library or procedure <*insert_3*> failed with error code <*insert_1*>.

Response
Either the library must be installed on the system or the environment changed to allow the program to locate it.

AMQ9221
Unsupported protocol was specified.

Severity
30 : Severe error

Explanation
The specified value of <*insert_3*> was not recognized as one of the protocols supported.

Response
Correct the parameter and retry the operation.

AMQ9222
Cannot find the configuration file.

Severity
10 : Warning

Explanation
The configuration file <*insert_3*> cannot be found. This file contains default definitions for communication parameters. Default values will be used.

Response
None.

AMQ9223
Enter a protocol type.

Severity
30 : Severe error

Explanation
The operation you are performing requires that you enter the type of protocol.

Response
Add the protocol parameter and retry the operation.

AMQ9224
Unexpected .ini file entry.

Severity
30 : Severe error

Explanation
.ini file keyword <*insert_3*> is either not a valid keyword or has an invalid value.

Response
Correct the file and retry the operation.

AMQ9224 (Windows)
Invalid registry value.

Severity
30 : Severe error

Explanation
WebSphere MQ registry value name <*insert_3*> is either not valid or has invalid value data.

Response
Correct the registry value and retry the operation.

AMQ9225
File syntax error.

Severity
30 : Severe error

Explanation
A syntax error was detected on line <*insert_1*> while processing the INI file.

Response
Correct the problem and retry the operation.

AMQ9225 (Windows)
File syntax error.

Severity
30 : Severe error

Explanation
A syntax error was detected while processing the configuration data.

Response
Correct the problem and retry the operation.

AMQ9226
Usage: <*insert_3*> [-m QMgrName] -t (TCP | LU62 | NETBIOS | SPX) [ProtocolOptions]

Severity
10 : Warning

Explanation
Values passed to the listener program were invalid.

The parameter string passed to this program is as follows:

[-m QMgrName] (-t TCP [-p Port] |

-t LU62 [-n TPName] |

-t NETBIOS [-l LocalName] [-e Names] [-s Sessions]

[-o Commands] [-a Adapter] |

-t SPX [-x Socket])

Default values will be used for parameters not supplied.

Response
Correct the parameters passed to the listener program and retry the operation.

AMQ9226 (AIX)
Usage: <*insert_3*> [-m QMgrName] -t TCP [ProtocolOptions]

Severity
10 : Warning

Explanation
Values passed to the listener program were invalid.

The parameter string passed to this program is as follows:

[-m QMgrName] -t TCP [-p Port]

Default values will be used for parameters not supplied.

Response
Correct the parameters passed to the listener program and retry the operation.

AMQ9226 (Unix)
Usage: <*insert_3*> [-m QMgrName] -t TCP [ProtocolOptions]

Severity
10 : Warning

Explanation
Values passed to the listener program were invalid.

The parameter string passed to this program is as follows:

[-m QMgrName] -t TCP [-p Port]

Default values will be used for parameters not supplied.

Response
Correct the parameters passed to the listener program and retry the operation.

AMQ9227
<*insert_3*> local host name not provided.

Severity
30 : Severe error

Explanation
A name is required for the <*insert_3*> process to register with the network.

Response
Add a local name to the configuration file and retry the operation.

AMQ9228
The <*insert_4*> responder program could not be started.

Severity
30 : Severe error

Explanation
An attempt was made to start an instance of the responder program, but the program was rejected.

Response
The failure could be because either the subsystem has not been started (in this case you should start the subsystem), or there are too many programs waiting (in this case you should try to start the responder program later). The <*insert_5*> reason code was <*insert_1*>.

AMQ9229
The application has been ended.

Severity
30 : Severe error

Explanation
You have issued a request to end the application.

Response
None.

AMQ9230
An unexpected <*insert_4*> event occurred.

Severity
30 : Severe error

Explanation
During the processing of network events, an unexpected event <*insert_1*> occurred.

Response
None.

AMQ9231
The supplied parameter is not valid.

Severity
30 : Severe error

Explanation
The value of the <*insert_4*> <*insert_5*> parameter has the value <*insert_3*>. This value has either not been specified or has been specified incorrectly.

Response
Check value of the <*insert_5*> parameter and correct it if necessary. If the fault persists, record the return code (<*insert_1*>,<*insert_2*>) and <*insert_4*> and tell the systems administrator.

AMQ9232
No <*insert_3*> specified

Severity
30 : Severe error

Explanation
The operation requires the specification of the <*insert_3*> field.

Response
Specify the <*insert_3*> and retry the operation.

AMQ9233
Error creating <*insert_3*> thread.

Severity
30 : Severe error

Explanation
The process attempted to create a new thread. The most likely cause of this problem is a shortage of an operating system resource (for example, memory). Use any previous FFSTs to determine the reason for the failure. The WebSphere MQ internal return code describing the reason for the failure is <*insert_1*>.

Response
Contact the systems administrator. If the problem persists save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9235
The supplied local communications address cannot be resolved.

Severity
30 : Severe error

Explanation
The local communications address (LOCLADDR) value <*insert_3*> cannot be resolved into an IP address.

Response
Enter a local communications address value which can be resolved into an IP address, and try again.

AMQ9236
The supplied Partner LU was invalid.

Severity
30 : Severe error

Explanation
The <*insert_4*> Partner LU name <*insert_3*> was invalid.

Response
Either the Partner LU name was entered incorrectly or it was not in the <*insert_4*> communications configuration. Correct the error and try again.

AMQ9237
A configuration error for <*insert_4*> occurred.

Severity
30 : Severe error

Explanation
Allocation of a <*insert_4*> conversation to host <*insert_3*> was not possible. The configuration error may be one of the following:

1. It may be that one of the transmission parameters (Mode, or TP Name) was incorrect. Correct the error and try again. The mode name should be the same as the mode defined on host <*insert_3*>. The TP name on <*insert_3*> should be defined.

2. It may be that an LU 6.2 session has not been established. Contact your systems administrator.

The return code from <*insert_4*> is <*insert_1*> with associated <*insert_5*> <*insert_2*> .

Response
Record the error values and tell the system administrator.

AMQ9238
A communications error for <*insert_4*> occurred.

Severity
30 : Severe error

Explanation
An unexpected error occurred in communications.

Response
The return code from the <*insert_4*> <*insert_3*> call was <*insert_1*> with associated <*insert_5*> <*insert_2*> .

AMQ9239
Usage: <*insert_3*> [-m QMgrName] -n TpName -g Gateway-name

Severity
10 : Warning

Explanation
Values passed to the listener program were invalid. The parameter string passed to this program is as follows, default values being used for parameters not supplied: [-m QMgrName] -n TpName -g Gateway-name

Response
Correct the parameters passed to the listener program and retry the operation.

AMQ9240
An SPX socket was already in use.

Severity
30 : Severe error

Explanation
The Listener received return code <*insert_1*> when attempting to open socket <*insert_2*>.

Response
The specified socket is already in use by another process. To use another socket specify another socket on the command line to RUNMQLSR or update the default in the qm.ini file.

AMQ9240 (Windows)
An SPX socket was already in use.

Severity
30 : Severe error

Explanation
The listener received return code <*insert_1*> when attempting to open socket <*insert_2*>.

Response
The specified socket is already in use by another process. To use another socket, specify a different socket on the command line to the runmqlsr command, or update the default in the configuration data.

AMQ9240 (IBM i)
An SPX socket was already in use.

Severity
30 : Severe error

Explanation
The Listener received return code <*insert_1*> when attempting to open socket <*insert_2*>.

Response
The specified socket is already in use by another process. To use another socket specify another socket on the command line to STRMQMLSR or update the default in the qm.ini file.

AMQ9241
SPX is not available.

Severity
30 : Severe error

Explanation
WebSphere MQ received return code <*insert_1*> when attempting to start SPX communications.

Response
Ensure that IPX/SPX support is installed on the machine and that it is started before trying to start a WebSphere MQ SPX channel.

AMQ9242
SPX resource problem.

Severity
30 : Severe error

Explanation
WebSphere MQ received return code <*insert_1*> when attempting to start SPX communications, indicating a resource problem.

Response
Ensure that sufficient IPX/SPX resources are available before commencing communications over IPX/SPX.

AMQ9243
The queue manager <*insert_3*> does not exist.

Severity
30 : Severe error

Explanation
You tried to perform an action against a queue manager that does not exist. You may have specified the wrong queue manager name.

Response
If you specified the wrong name, correct the name and submit the command again. If the queue manager does not exist, create the queue manager and submit the command again.

AMQ9244
The default queue manager does not exist.

Severity
30 : Severe error

Explanation
You tried to perform an action against a queue manager that does not exist.

Response
Create the default queue manager and submit the command again.

AMQ9245 (Windows)
Unable to obtain account details for channel MCA user ID.

Severity
10 : Warning

Explanation
WebSphere MQ was unable to obtain the account details for MCA user ID <*insert_3*>. This user ID was the MCA user ID for channel <*insert_4*> on queue manager <*insert_5*> and may have been defined in the channel definition, or supplied either by a channel exit or by a client.

Response
Ensure that the user ID is correct and that it is defined on the Windows local system, the local domain or on a trusted domain. For a domain user ID, ensure that all necessary domain controllers are available.

AMQ9246
The TCP/IP listener on port <*insert_1*> could not start a new channel.

Severity
30 : Severe error

Explanation
An attempt has been made to connect to the queue manager by starting a new channel within the TCP/IP listener which is listening on port <*insert_1*>. The maximum socket number which can be used by a channel running on this listener is <*insert_2*> . A socket number beyond this maximum was allocated for the new channel. This connection attempt has been rejected, but the listener continues to listen for further connection requests. The socket number allocated for a new listener channel is related to the number of channels currently running within that listener process. The problem has arisen because too many channels are directed at the port on which this listener is listening.

Response
An extra listener process should be started to listen on a different port. Some of the channels to the queue manager should be redirected from the port on which the existing listener is listening to the new port.

AMQ9247
SSPI Security: bad return from SSPI call.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> has been closed because the SSPI channel exit received a bad return code from SSPI.

Response
Consult the appropriate SSPI manuals to find out the meaning of status <*insert_4*> on call <*insert_5*> , and correct the error.

AMQ9248
The program could not bind to a <*insert_3*> socket.

Severity
30 : Severe error

Explanation
The attempt to bind to socket <*insert_4*> failed with return code <*insert_1*>. The failing <*insert_3*> call was <*insert_5*> . The most likely cause of this problem is incorrect configuration of the <*insert_3*> local address or incorrect start and end port parameters.

Response
Contact the system administrator. If the problem persists save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9255
Listener already running.

Severity
30 : Severe error

Explanation
The request to start the WebSphere MQ listener failed because there is already a listener running against the specified network resources.

Response
None.

AMQ9259
Connection timed out from host <*insert_3*> .

Severity
30 : Severe error

Explanation
A connection from host <*insert_3*> over <*insert_4*> timed out.

Response
Check to see why data was not received in the expected time. Correct the problem. Reconnect the channel, or wait for a retrying channel to reconnect itself.

AMQ9262 (HP-UX)
GSKit SSL support not available for 32-bit client applications.

Severity
20 : Error

Explanation
An attempt was made to start an SSL channel from a 32-bit client application. However, GSKit SSL 32-bit support is not provided on WebSphere MQ for HP-UX (Itanium platform).

Response

Compile the client application as a 64-bit application or change the application to use a non-SSL channel.

AMQ9268 (rrcI_SCTQ_SWITCH_SUCCESS)

Cluster sender channel <*insert one*> successfully switched to use the transmission queue <*insert three*> .

Severity

00 : Information

Explanation

The transmission queue for cluster sender channel <*insert one*> was successfully switched from <*insert two*> to <*insert three*>.

Response

None.

AMQ9270

Sharing conversation could not start.

Severity

30 : Severe error

Explanation

The attempt to start sharing conversation <*insert_1*> on socket <*insert_2*> (channel <*insert_3*>) was rejected at the server-connection end of the channel.

Response

Examine diagnostic information at the server-connection end of channel <*insert_3*> to see why the conversation did not start. If possible, correct the error causing the failure and retry.

AMQ9271

Channel <*insert_3*> timed out.

Severity

30 : Severe error

Explanation

A timeout occurred while waiting to receive from the other end of channel <*insert_3*>. The address of the remote end of the connection was <*insert_4*>.

Response

The return code from the <*insert_5*> call was <*insert_1*> (X<*insert_2*>). Record these values and tell the systems administrator.

AMQ9272

Thread mutex semaphore error.

Severity

30 : Severe error

Explanation

The process attempted an operation on a thread mutex semaphore. The most likely cause of this problem is a shortage of an operating system resource (for example, memory). Use any previous FFSTs to determine the reason for the failure. The WebSphere MQ function involded was <*insert_3*> and the internal return code describing the reason for the failure is <*insert_1*>.

Response

Contact the systems administrator. If the problem persists save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9273
Thread event error.

Severity
30 : Severe error

Explanation
The process attempted an operation on a thread event. The most likely cause of this problem is a shortage of an operating system resource (for example, memory). Use any previous FFSTs to determine the reason for the failure. The WebSphere MQ function involved was <*insert_3*> and the internal return code describing the reason for the failure is <*insert_1*>.

Response
Contact the systems administrator. If the problem persists save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9280 (rrcE_SSL_SUITE_B_INVALID_VALUE)
Parameter requesting Suite B contains an invalid value.

Severity
30 : Severe error

Explanation
An SSL or TLS channel running on an WebSphere MQ client has failed to start. This is because the MQSUITEB environment variable, or the MQSCO EncryptionPolicySuiteBStrength field, contains an invalid value. The values specified were ' <*insert_1*>'.

The channel is '<*insert_2*>', in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Set the MQSUITEB environment variable, or the MQSCO EncryptionPolicySuiteBStrength field to a valid value.

Restart the channel.

Refer to the WebSphere MQ Security documentation for more information on Suite B configuration.

AMQ9281 (rrcE_SSL_SUITE_B_BAD_COMBINATION)
Parameter requesting Suite B contains an invalid combination of values.

Severity
30 : Severe error

Explanation
An SSL or TLS channel running on an MQ client has failed to start. This is because the MQSUITEB environment variable, or the MQSCO EncryptionPolicySuiteBStrength field, contain mutually exclusive values. All of the values are valid, but some of them cannot be used together. The values specified were ' <*insert_1*>'

The channel is '<*insert_1*>', in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Set the MQSUITEB environment variable, or the MQSCO EncryptionPolicySuiteBStrength field, to a valid combination of values

Restart the channel.

Refer to the WebSphere MQ Security documentation for more information on Suite B configuration.

AMQ9282 (rrcE_SSL_CIPHER_INVALID_SUITE_B)
Invalid CipherSpec for the configured Suite B security level.

Severity
30 : Severe error

Explanation
The user is attempting to start a channel on a queue manager or WebSphere MQ client which has been configured to run in Suite B mode. The user has specified a CipherSpec which does not meet the configured Suite B security level.

The channel is '*<insert_1>*', in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

The address of the remote host is '*<insert_2>* '.

Response
Redefine the channel to run with a Suite B compliant CipherSpec which satisfies the configured Suite B security level. Alternatively, the channel may be defined with the correct CipherSpec and the queue manager or WebSphere MQ client should not be running in Suite B mode; if this is the case, ensure that Suite B mode is not configured. Once the error is corrected, restart the channel.

Refer to the WebSphere MQ Security documentation for more information on Suite B security levels or CipherSpecs.

This message might occur after applying WebSphere MQ maintenance because the FIPS and Suite B standards are updated periodically. When such changes occur, WebSphere MQ is also updated to implement the latest standard. As a result, you might see changes in behavior after applying maintenance. For more information about the versions of FIPS and Suite B standards enforced by WebSphere MQ, see the readme file.

AMQ9285 (rrcE_SSL_CIPHER_AND_CERT_INCOMPATIBLE)
The proposed CipherSpec is incompatible with a digital certificate.

Severity
30 : Severe error

Explanation
The SSL or TLS handshake failed because the proposed CipherSpec is incompatible with one of the digital certificates.

It is necessary for both the local and remote systems to use a digital certificate which is suitable for use with the channel CipherSpec. Common causes of this error include:

(a) An RSA-based CipherSpec was specified when using a certificate which contains a non-RSA public key.

(b) An Elliptic Curve-based CipherSpec was specified when using a certificate which contains a non-EC public key.

The channel is '*<insert_1>*', in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Specify a different CipherSpec which is suitable for use with the digital certificates used on both the local and remote systems. Restart the channel.

Refer to the WebSphere MQ Security documentation for further information on CipherSpecs.

AMQ9289 (rrcI_SCTQ_MSGMOVE_NONE)
Message move complete - no messages moved.

Severity
00 : Information

Explanation
No messages were moved whilst switching the transmission queue for cluster sender channel <*insert one*>. The message move operation is complete.

Response
None.

AMQ9290 (rrcI_SCTQ_MSGMOVE_IN_PROGRESS)
Message move in progress - <*n*> messages moved.

Severity
00 : Information

Explanation
<*n*> messages were moved whilst switching the transmission queue for cluster sender channel <*insert one*>. The message move operation is in progress.

Response
None.

AMQ9291 (rrcI_SCTQ_MSGMOVE_COMPLETE)
Message move complete - <*n*> messages moved.

Severity
00 : Information

Explanation
<*n*> messages were moved whilst switching the transmission queue for cluster sender channel <*insert one*>. The message move operation is complete.

Response
None.

AMQ9301 (Tandem)
An SNA communications error occurred.

Severity
30 : Severe error

Explanation
An unexpected error occurred in communications.

Response
The reply return code from the SNAX/ICE <*insert_3*> request was <*insert_1*> in the <*insert_4*> header. The detail return code was <*insert_2*> .

AMQ9302 (Tandem)
The TCP Listener <*insert_3*> in Queue Manager <*insert_4*> cannot find an available port.

Severity
40 : Stop Error

Explanation
The TCP Listener has tried all the ports that are configured in the QMINI file for this Queue Manager, and none were available for listening on. The TCP Listener has now terminated. The TCP Listener is either not needed (because there are already TCP Listeners running on all the Queue Manager ports), or there is a configuration problem with the Queue Manager.

Response
Review the QMINI file TCP/IP Listener stanzas to determine if there is a configuration problem. The ports numbers themselves may be incorrect, or overlap with the ports being used by other Queue Managers on the same system, or with other services.

AMQ9401
Channel <*insert_3*> autodefined.

Severity
0 : Information

Explanation
Channel <*insert_3*> which did not previously exist has been autodefined.

Response
None.

AMQ9402
Autodefinition exit for Channel <*insert_3*> failed to load.

Severity
30 : Severe error

Explanation
Autodefinition of Channel <*insert_3*> failed because <*insert_4*> would not load.

Response
Ensure that the user exit is specified correctly in the queue manager definition, and that the user exit program is correct and available.

AMQ9403
Autodefinition of Channel <*insert_3*> suppressed by user exit.

Severity
30 : Severe error

Explanation
Autodefinition exit <*insert_4*> for Channel <*insert_3*> returned a failure code.

Response
None.

AMQ9404
Phase one of REFRESH CLUSTER REPOS(YES) has completed.

Severity
0 : Information

Explanation
Phase one of REFRESH CLUSTER REPOS(YES) has completed. The Refresh Cluster(<*insert_4*>) command caused <*insert_1*> objects to be refreshed and republished to <*insert_2*> queue managers. Applications attempting to access cluster resources may see failures to resolve cluster resources until phase two of REFRESH CLUSTER is complete. Phase two is complete when all new information has been received from other members of the cluster.

Response
Monitor your SYSTEM.CLUSTER.COMMAND.QUEUE to determine when it has reached a consistently empty state to indicate that the refresh process has completed.

AMQ9405
FORCEREMOVE QUEUES(YES) command processed, cluster <*insert_3*> target <*insert_4*>.

Severity
0 : Information

Explanation
The repository queue manager successfully processed a RESET ACTION(FORCEREMOVE) command with the QUEUES(YES) option for the indicated cluster and target queue manager.

Response
None.

AMQ9406
REFRESH CLUSTER REPOS(YES) command failed, this queue manager is a full repository for cluster <*insert_4*>.

Severity
30 : Severe error

Explanation
The repository queue manager could not process a REFRESH CLUSTER command with the REPOS(YES) option for the indicated cluster, because the local queue manager provides full repository management services for the cluster. The command is ignored.

Response
Either

1) Reissue the command without REPOS(YES), or

2) Issue the command on a queue manager which is not a full repository, or

3) Change this queue manager definition so that it is not a full repository.

AMQ9407
Cluster queue <*insert_3*> is defined inconsistently.

Severity
10 : Warning

Explanation
The definition of cluster queue <*insert_3*> on the queue manager with UUID <*insert_4*> has different DEFPRTY, DEFPSIST and DEFBIND values from the definition of the same cluster queue on the queue manager with UUID <*insert_5*>. Both definitions now exist in the local repository. All definitions of the same cluster queue should be identical. In particular, problems arise if your applications rely on a queue default value which is defined inconsistently to determine messaging behavior. This applies, for example, if the applications open a cluster queue with option MQOO_BIND_AS_Q_DEF. If different instances of the queue have different DEFBIND values the behavior of the message transfer differs depending on which instance of the queue is selected when it is opened. In general the instance selected varies across opens.

Response
For each inconsistency decide which of the values is the correct one. Alter the definitions of cluster queue <*insert_3*> so that all definitions have correct DEFPRTY, DEFPSIST and DEFBIND values.

AMQ9408
BIND_ON_OPEN messages for channel <*insert_3*> to dead-letter queue.

Severity
0 : Information

Explanation
The remote CLUSRCVR for channel <*insert_3*> was deleted while undelivered BIND_ON_OPEN messages associated with that channel existed on the local SYSTEM.CLUSTER.TRANSMIT.QUEUE. These messages could not be allocated to another channel because they were put BIND_ON_OPEN, but were very unlikely to ever flow along the channel with which they were associated as this has now been deleted. An attempt has therefore been made to move them from the transmission queue to the local dead-letter queue. The MQDLH reason is MQFB_BIND_OPEN_CLUSRCVR_DEL. Note that any internal WebSphere MQ Clustering messages for the deleted channel will also have been removed from the SYSTEM.CLUSTER.TRANSMIT.QUEUE (these are discarded) so the current depth of the queue may have decreased by more than the number of user messages moved to the dead-letter queue.

Response

Examine the contents of the dead-letter queue. Each message is contained in an MQDLH structure that includes the reason why it was written and where it was originally addressed. Also look at previous error messages to see if the attempt to put messages to the dead-letter queue failed.

AMQ9409

Repository manager ended abnormally.

Severity

30 : Severe error

Explanation

The repository manager process ended abnormally. Termination of this process will cause the queue manager to terminate unless the tuning parameter TolerateRepositoryFailure has been set to 'TRUE'. If the queue manager does not terminate, further cluster management activity will not occur, this will effect the availability of cluster resources accessed or hosted by this queue manager.

Response

Look at previous error messages for the repository manager in the queue manager and system error logs to determine the cause of the failure or contact your IBM support center. Restart the queue manager to restart the repository manager process.

AMQ9410

Repository manager started

Severity

0 : Information

Explanation

The repository manager started successfully.

Response

None.

AMQ9411

Repository manager ended normally.

Severity

0 : Information

Explanation

The repository manager ended normally.

Response

None.

AMQ9412

Repository command received for <*insert_3*> .

Severity

30 : Severe error

Explanation

The repository manager received a command intended for some other queue manager, with identifier that is <*insert_3*> . The command was sent by the queue manager with identifier <*insert_4*>.

Response

Check the channel and cluster definitions of the sending queue manager.

AMQ9413

Repository command format error, command code <*insert_1*>

Severity
30 : Severe error

Explanation
An internal error has occurred.

Response
Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9415
Repository command unexpected, command code <*insert_1*> , cluster object <*insert_3*>, sender <*insert_4*>

Severity
30 : Severe error

Explanation
An internal error has occurred.

Response
Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9415 (IBM i)
An internal error has occurred.

Severity
30 : Severe error

Explanation
Repository command unexpected, command code <*insert_1*> , cluster object <*insert_3*>, sender <*insert_4*>

Response
Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9416
Repository command processing error, RC=<*insert_2*> , command code <*insert_1*>, cluster object <*insert_3*>, sender <*insert_4*>.

Severity
30 : Severe error

Explanation
An internal error has occurred.

Response
Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9416 (IBM i)

An internal error has occurred.

Severity

30 : Severe error

Explanation

Repository command processing error, RC=<*insert_2*> , command code <*insert_1*>, cluster object <*insert_3*>, sender <*insert_4*>.

Response

Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9417

Manually defined CLUSSDR channels have been forcibly removed.

Severity

0 : Information

Explanation

The administrator has asked for the queue manager <*insert_3*> to be deleted, or forcibly removed, but has not yet deleted the manually defined CLUSSDR channels to <*insert_3*> . The auto-defined channels to <*insert_3*> have been deleted, but <*insert_3*> continues to receive updates until the manually defined CLUSSDR channels have been deleted.

Response

Delete the manually defined CLUSSDR channels to <*insert_3*> .

AMQ9418

Only one repository for cluster <*insert_3*> .

Severity

0 : Information

Explanation

The queue manager has received information about a cluster for which it is the only repository.

Response

Alter the REPOS or REPOSNL attribute of the queue manager, that is to have the second full repository for the cluster, to specify the cluster name.

AMQ9419

No cluster-receiver channels for cluster <*insert_3*>

Severity

0 : Information

Explanation

The repository manager has received information about a cluster for which no cluster-receiver channels are known.

Response

Define cluster-receiver channels for the cluster on the local queue manager.

AMQ9420

No repositories for cluster <*insert_3*>.

Severity

0 : Information

Explanation

The queue manager has received information about a cluster for which no repositories are known.

Response

Alter the REPOS or REPOSNL attribute of the queue manager, that is to have a full repository for the cluster, to specify the cluster name.

AMQ9421

Invalid cluster record action code detected

Severity

30 : Severe error

Explanation

An invalid record was read from the SYSTEM.CLUSTER.REPOSITORY.QUEUE. An FFST record has been generated containing the invalid record.

Response

Collect the items listed in the Problem Determination section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9422

Repository manager error, RC=<*insert_1*>

Severity

30 : Severe error

Explanation

An internal error has occurred.

Response

Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9425

An internal error has occurred.

Severity

30 : Severe error

Explanation

Repository command merge error, command code <*insert_1*> , cluster object <*insert_3*>, sender <*insert_4*>

Response

Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9426

Repository command recipient unknown.

Severity

30 : Severe error

Explanation

The repository manager tried to send a command to another queue manager using channel <*insert_4*>. The recipient queue manager, with an identifier that is <*insert_3*> , could not be found. Command code <*insert_1*> .

Response

Check the channel and cluster definitions of the sending and receiving queue managers.

AMQ9427

CLUSSDR channel does not point to a repository queue manager.

Severity

30 : Severe error

Explanation

A CLUSSDR channel must point to a queue manager that hosts repositories for all clusters of which the channel is a member. In addition, the CLUSRCVR for the channel must be a member of all the same clusters as the CLUSSDR channel. The queue manager pointed to by CLUSSDR channel <*insert_3*> does not meet these criteria for cluster <*insert_4*>. The remote queue manager has a QMID of <*insert_5*>.

Response

Check the definitions on the local and remote queue managers to ensure that the CLUSSDR channel points to a queue manager that hosts a repository for the cluster, and that the CLUSRCVR for the channel is a member of the cluster.

AMQ9428

Unexpected publication of a cluster queue object received.

Severity

30 : Severe error

Explanation

The local queue manager has received a publication of a cluster queue object from a remote queue manager on cluster <*insert_3*> . The local queue manager discards the request because it does not host a repository for cluster <*insert_3*> and has not subscribed to the published object. The remote CLUSSDR channel used to access the local queue manager has a channel name of <*insert_4*> and the remote queue manager has a QMID of <*insert_5*>.

Response

Check the definitions on the local and remote queue managers to ensure that the CLUSSDR channel points to a repository queue manager for the cluster.

AMQ9429

Unexpected publication of a cluster queue deletion received.

Severity

30 : Severe error

Explanation

The local queue manager has received a publication of a cluster queue deletion from a remote queue manager on cluster <*insert_3*> . The local queue manager discards the request because it does not host a repository for cluster <*insert_3*> and has not subscribed to the published object. The remote CLUSSDR channel used to access the local queue manager has a channel name of <*insert_4*> and the remote queue manager has a QMID of <*insert_5*>.

Response

Check the definitions on the local and remote queue managers to ensure that the CLUSSDR channel points to a repository queue manager for the cluster.

AMQ9430

Unexpected cluster queue manager publication received.

Severity
30 : Severe error

Explanation
The local queue manager has received a cluster queue manager publication on cluster <*insert_3*>. The local queue manager should not have received the publication because it does not host a repository for cluster <*insert_3*> , it has not subscribed to information concerning the published object, and the published object does not match any of its CLUSSDRs. The queue manager that sent the publication to the local queue manager has QMID <*insert_4*> (note that this is not necessarily the queue manager which originated the publication). CLUSSDR channel <*insert_5*> was used to send the publication.

Response
Check the CLUSSDR definition on the sending queue manager to ensure that it points to a repository queue manager for the cluster.

AMQ9431
Remote queue manager no longer hosts a repository for cluster

Severity
0 : Information

Explanation
The local queue manager has received a message from remote queue manager QMID <*insert_3*> indicating that it no longer hosts a repository for cluster <*insert_4*> . CLUSSDR channel <*insert_5*> is altered so that it can no longer be used to access queue manager <*insert_3*> within cluster <*insert_4*>. If the local queue manager does not host a repository for cluster <*insert_4*> the relevant subscriptions and publications are remade if possible.

Response
None.

AMQ9432
Query received by a non-repository queue manager

Severity
30 : Severe error

Explanation
The local queue manager has received a query from a remote queue manager on cluster <*insert_3*>. The local queue manager discards the query because it does not host a repository for cluster <*insert_3*>. The remote CLUSSDR channel used to access the local queue manager has a channel name of <*insert_4*> and the remote queue manager has a QMID of <*insert_5*>.

Response
Check the definitions on the local and remote queue managers to ensure that the CLUSSDR channel points to a repository queue manager for the cluster.

AMQ9433
CLUSRCVR must be in the same cluster as its matching CLUSSDR.

Severity
30 : Severe error

Explanation
CLUSRCVR channel <*insert_3*> is not defined as a member of cluster <*insert_4*>. The local queue manager has received a command that indicates that CLUSSDR channel <*insert_3*> on the remote queue manager with QMID <*insert_5*> is defined as a member of cluster <*insert_4*>.

Response
Alter the CLUSRCVR or CLUSSDR definitions for channel <*insert_3*>, so that they are both members of the same cluster.

AMQ9434

Unrecognized message on <*insert_3*>.

Severity

30 : Severe error

Explanation

The repository manager found a message on one of its queues having, either a format that could not be recognized, or that did not come from a queue manager or repository manager. The message was put on the dead-letter queue.

Response

Examine the message on the dead-letter queue to determine the originator of the message.

AMQ9435

Unable to put repository manager message.

Severity

30 : Severe error

Explanation

The repository manager tried to send a message to the SYSTEM.CLUSTER.COMMAND.QUEUE on another queue manager with an identifier that is <*insert_3*> , but the MQPUT call was unsuccessful. MQCC=<*insert_1*> , MQRC=<*insert_2*>. Processing continues, but the repository information may be out of date.

Response

Refer to the Application Programming Reference manual for information about MQCC <*insert_1*> and MQRC <*insert_2*> . Check the channel and cluster definitions on the local and target queue managers, and ensure that the channels between them are running. When the problem is corrected, the repository information will normally be updated automatically. The REFRESH CLUSTER command can be used to ensure that the repository information is up to date.

AMQ9436

Unable to send repository manager message.

Severity

30 : Severe error

Explanation

The repository manager tried to send a message to the SYSTEM.CLUSTER.COMMAND.QUEUE on a queue manager that has the full repository for the specified cluster(<*insert_3*>), but the MQPUT call was unsuccessful. MQCC=<*insert_1*>, MQRC=<*insert_2*>. Processing continues, but repository information may be out of date.

Response

Refer to the Application Programming Reference manual for information about MQCC <*insert_1*> and MQRC <*insert_2*> . Check the channel and cluster definitions on the local and target queue managers, and ensure that the channels between them are running. When the problem is corrected, the repository information will normally be updated automatically. The REFRESH CLUSTER command can be used to ensure that the repository information is up to date.

AMQ9437

Unable to commit repository manager changes.

Severity

30 : Severe error

Explanation

The repository manager tried to commit some internal operations but was unsuccessful. The reason code from the MQCMIT call was <*insert_1*>

Response

Inspect the reason code. If it does not seem reasonable in the context of the other queue manager operations taking place at the time, then save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9438

CONNAME could not be discovered for CLUSRCVR <*insert_3*> .

Severity

30 : Severe error

Explanation

TCP/IP CLUSRCVR <*insert_3*> was validly specified with a blank or absent CONNAME parameter. However when the repository process, amqrrmfa, attempted to obtain the CONNAME (IP address) for itself it was unable to. If there is an existing matching CLUSRCVR object in the cache its CONNAME is used. The CONNAME used was <*insert_4*>.

Response

Check the error log for a message arising from an associated TCP/IP call (gethostname, gethostbyname or inet_ntoa). Pass all the error information to your systems administrator.

AMQ9439

Repository corruption: bad CLQMGR object for channel <*insert_3*>.

Severity

30 : Severe error

Explanation

An internal error has occurred.

Response

Collect the items listed in Problem determination and use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9440

Reset command failed.

Severity

0 : Information

Explanation

Reset Cluster(<*insert_3*>) Qmname(<*insert_4*>) command failed. To issue this command, queue manager <*insert_5*> must be a repository for cluster <*insert_3*>. Alter the queue manager attributes Repos, or Reposnl, to include cluster <*insert_3*> and retry the command.

Response

None.

AMQ9441

Reset command processed.

Severity

0 : Information

Explanation

The reset Cluster(<*insert_3*>) Qmname(<*insert_4*>) command has processed on this repository and <*insert_1*> other queue managers have been sent notification.

Response
None.

AMQ9442
Phase one of the REFRESH CLUSTER command has completed.

Severity
0 : Information

Explanation
Phase one of the REFRESH CLUSTER command has completed. The Refresh Cluster(<*insert_4*>) command caused <*insert_1*> objects to be refreshed, and republished to <*insert_2*> queue managers.

Applications attempting to access cluster resources might see failures to resolve cluster resources until phase two of REFRESH CLUSTER is complete. Phase two is complete when all new information has been received from other members of the cluster.

Response
Monitor your SYSTEM.CLUSTER.COMMAND.QUEUE to determine when it has reached a consistently empty state to indicate that the refresh process has completed.

AMQ9443
Suspend Qmgr Cluster command processed.

Severity
0 : Information

Explanation
The Suspend Qmgr Cluster command completed. <*insert_1*> objects suspended.I n the case of a name list the cluster name is the first name in the list.

Response
None.

AMQ9444
Resume Qmgr Cluster command processed.

Severity
0 : Information

Explanation
The Resume Qmgr Cluster(<*insert_4*>) command completed. <*insert_1*> objects resumed. In the case of a name list the cluster name is the first name in the list.

Response
None.

AMQ9445
Error creating channel <*insert_3*>.

Severity
30 : Severe error

Explanation
Channel <*insert_4*> tried to replace itself by creating channel <*insert_3*>. The attempt to create the channel was unsuccessful for the following reason: " <*insert_5*>". A previous message may give further information.

Response
Rectify the problem which prevented successful creation of channel <*insert_3*>. Restart channel <*insert_4*> .

AMQ9446
Error deleting channel <*insert_3*>.

Severity

30 : Severe error

Explanation

Channel <*insert_3*> tried to delete itself after creating channel <*insert_4*> to replace it. The attempt to delete the channel was unsuccessful for the following reason: "<*insert_5*>".

Response

If channel <*insert_3*> still exists rectify the problem which prevented its deletion and then manually delete the channel.

AMQ9447

Unable to backout repository manager changes.

Severity

30 : Severe error

Explanation

The repository manager tried to back out some internal operations but was unsuccessful. The reason code from the MQBACK call was <*insert_1*>.

Response

Inspect the reason code. If it does not seem reasonable in the context of the other queue manager operations taking place at the time, then save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9448

Repository manager failed due to errors. Retry in <*insert_1*> minutes, queue manager will terminate in <*insert_2*> minutes.

Severity

30 : Severe error

Explanation

Repository manager encountered a severe problem. See the earlier messages in the queue manager or system error logs for details. The repository manager will retry the command in <*insert_1*> minutes. If the problem is not rectified in <*insert_2*> minutes the queue manager will terminate. Until this problem is rectified no further cluster management activity will occur, this will effect the availability of cluster resources accessed or hosted by this queue manager.

Response

If possible, rectify the identified problem, otherwise contact your IBM support center. To postpone the queue manager from terminating due to this problem set the SYSTEM.CLUSTER.COMMAND.QUEUE queue to be GET(DISABLED). Once the problem has been rectified, set the queue to be GET(ENABLED) and wait for the repository manager to retry the command or restart the queue manager.

AMQ9449

The repository manager is restarting following an error.

Severity

0 : Information

Explanation

The repository manager is restarting following an error, see earlier error messages for details of the failure.

Response

If the failure re-occurs contact your IBM support center and follow any instructions in the subsequent error messages.

AMQ9450

Usage: <*insert_3*> [-m QMgrName] -f OutputFile [-v OutputFileVersion]

Severity

10 : Warning

Explanation

Values passed to the channel table writer program were invalid.

The parameter string passed to this program is as follows:

[-m QMgrName] -f OutputFile [-v OutputFileVersion]

where OutputFileVersion can be either 2 or 5 (5 is the default)

Default values will be used for parameters not supplied.

Response

Correct the parameters passed to the channel table writer program and retry the operation.

AMQ9451 (Tandem)

Repository already active in CPU <*insert_1*>

Severity

0 : Information

Explanation

During initialization, a Repository Manager determined that the named CPU already had an active Repository Manager. This is probably caused by an incorrectly configured Pathway. Each CPU can support only one active Repository Manager.

Response

Ensure Pathway configuration only defines one Repository Manager per CPU

AMQ9453

FORCEREMOVE command failed, cluster <*insert_3*> target <*insert_4*> is not unique.

Severity

0 : Information

Explanation

The repository queue manager could not process a RESET ACTION(FORCEREMOVE) command for the indicated cluster and target queue manager, because there is more than one queue manager with the specified name in the cluster. The command is ignored.

Response

Reissue the command specifying the identifier (QMID) of the queue manager to be removed, rather than its name.

AMQ9453 (Tandem)

Repository Manager (CPU <*insert_1*>) partner in CPU <*insert_2*> closed

Severity

0 : Information

Explanation

The Repository Manager running in the first-named CPU noticed that a partner Repository Manager in the second-named CPU ended. This may be the result of the Queue Manager shutting down or it may indicate that the partner Repository Manager was forcibly stopped or suffered an error.

Response

If the Queue Manager is shutting down, this message is informational only. Otherwise, the WebSphere MQ error log, the system log, or both should be examined to determine why the partner Repository Manager ended.

AMQ9455
FORCEREMOVE command failed, cluster <*insert_3*> , target <*insert_4*>, not found.

Severity
0 : Information

Explanation
The repository queue manager could not process a RESET ACTION(FORCEREMOVE) command for the indicated cluster and target queue manager, because no information about that queue manager was found in the local repository. The command is ignored.

Response
Reissue the command, specifying the correct queue manager name or identifier.

AMQ9456
Update not received for queue <*insert_3*> , queue manager <*insert_4*> from full repository for cluster <*insert_5*>.

Severity
0 : Information

Explanation
The repository manager detected a queue that has been used in the last 30 days for which updated information should have been sent from a full repository. However, this has not occurred.

The repository manager will keep the information about this queue for a further 60 days.

Response
If the queue is still required, check that:

1) The cluster channels to and from the full repository and the queue manager that hosts the queue, are able to run.

2) The repository managers running on these queue managers have not ended abnormally.

AMQ9457
Repository available, cluster <*insert_4*> , channel <*insert_5*>, sender <*insert_3*> .

Severity
0 : Information

Explanation
The repository queue manager received a command from another queue manager, with an identifier that is <*insert_3*> , reporting that it is again a repository for cluster <*insert_4*> . The cluster-sender channel <*insert_5*> is changed so that it can be used to access the other queue manager in relation to the cluster.

Response
None.

AMQ9458
Unable to access the repository cache exclusively.

Severity
30 : Severe error

Explanation
A process remains registered as requiring access to the repository cache during an operation that must have exclusive access to the cache. The queue manager <*insert_3*> issues this message after waiting for the process to remove its registration, but the registration is still present. The process preventing exclusive access to the repository cache has <*insert_2*> outstanding registrations.

Response
The registered process identifier (PID) accessing the repository cache is <*insert_1*>. Determine if

this process is still running or terminated. If the process is not running or if the problem persists collect the items listed in the 'Problem determination' section of the System Administration manual and contact your IBM support center.

AMQ9459

Cluster topic <*insert_3*> from <*insert_4*> rejected due to PSCLUS(DISABLED).

Severity

10 : Warning

Explanation

Queue manager attribute PSCLUS has been set to DISABLED to indicate that inter queue manager Publish/Subscribe activity is not expected in this cluster. However, information regarding cluster topic <*insert_3*> has been sent to this queue manager over a channel from <*insert_4*>. The cluster topic definition is ignored and will not be visible from this queue manager.

Response

If you need to enable publish/subscribe clustering, alter the PSCLUS attribute on all queue managers in the cluster to ENABLED. You might also need to issue REFRESH CLUSTER and REFRESH QMGR commands as detailed in the PSCLUS documentation. If you are not using publish/subscribe clusters you should delete the clustered topic object, and ensure that PSCLUS is DISABLED on all queue managers.

AMQ9465

New cluster topic definition inconsistent.

Severity

10 : Warning

Explanation

The definition of cluster topic <*insert_3*> on the queue manager with UUID <*insert_4*> has a different <*insert_5*> attribute value than one or more cluster topics that already exist in the cluster cache. The existing topic objects are reported by message AMQ9466. All definitions of the same cluster topic should be identical, otherwise, problems may arise if your applications rely on one of these attributes to determine messaging behavior. For example, if an application opens a cluster topic and the different instances of the topic have different TOPICSTR values, the behavior of the message transfer depends on which instance of the topic happens to be selected when it is opened.

Response

Alter the definitions of the topic on the various queue managers so that they have identical values for all attributes.

AMQ9466

Cluster topic definitions inconsistent.

Severity

10 : Warning

Explanation

The definition of cluster topic <*insert_3*> on the queue manager with UUID <*insert_4*> has a different <*insert_5*> attribute value than a cluster topic being added to the cluster cache. The topic object being added is reported by message AMQ9465. All definitions of the same cluster topic should be identical, otherwise, problems may arise if your applications rely on one of these attributes to determine messaging behavior. For example, if an application opens a cluster topic and the different instances of the topic have different TOPICSTR values, the behavior of the message transfer depends on which instance of the topic happens to be selected when it is opened.

Response

Alter the definitions of the topic on the various queue managers so that they have identical values for all attributes.

AMQ9467

Repository error updating topic.

Severity

20 : Error

Explanation

The cluster repository manager received an unexpected error code from the queue manager while updating topic <*insert_3*> . The completion code was <*insert_1*>, reason code was <*insert_2*>.

Response

Refer to the WebSphere MQ product documentation for information about reason code <*insert_1*>.

Collect the items listed in the *Problem determination* section of the WebSphere MQ product documentation, and use either the WebSphere MQ Support site: , or the IBM Support Assistant (ISA): , to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9468

Cluster receiver channel <*insert_3*> has been configured by multiple queue managers.

Severity

0 : Informational

Explanation

Queue manager <*insert_4*> has joined a cluster using a cluster receiver channel with the same name as one that has already been defined by queue manager <*insert_5*> . All cluster receiver channels used within a cluster must be uniquely named. Only the last queue manager to join the cluster will use the named channel, Queue manager <*insert_5*> will not successfully participate in the cluster while the newer queue manager is a member.

Response

The use of a channel name currently associated with a different queue manager in the cluster may be intentional, for example, the original queue manager may have been deleted and re-created as a new queue manager. However, accidental duplication of a channel name across multiple queue managers will also result in this behaviour. If this was not intended, further investigation into the configuration of the queue managers should be performed.

AMQ9469

Update not received for CLUSRCVR channel &3 hosted on queue manager &4 in cluster &5.

Severity

10 : Warning

Explanation

The repository manager detected that the CLUSRCVR channel has not been republished by its owning queue manager. This republish action should have happened automatically <*insert_1*> between then and now.

The repository manager will check for this condition approximately every hour, continuing for a period of approximately <*insert_2*> days from now. If an update for the CLUSRCVR channel is received during this period, these messages will stop. If no update is received, these messages will continue to be written. However, after this period has elapsed, if no update has been received, the local queue manager will discard its knowledge of this channel, and these messages will stop. You should be aware that partial repository queue managers in this cluster will cease to be able to use the channel at about that time.

Response

There are several possible responses:

1. If the channel had been removed intentionally, and is no longer required, you should consider removing it fully by using the RESET CLUSTER command.
2. There is a long-running problem with the local queue manager's CLUSRCVR in cluster <*insert_5*>. If this is true, then correct the problem urgently to ensure that updates for the cluster are received.
3. There is a long-running problem on the remote queue manager's CLUSSDR in cluster <*insert_5*>. If this is true, then correct the problem urgently to ensure that updates for the cluster are sent.
4. Check that the repository manager on the remote queue manager has not ended abnormally.
5. The remote queue manager is out of step with this queue manager, potentially due to a restore of the queue manager from a backup. The remote queue manager must issue REFRESH CLUSTER to synchronize with other queue managers in the cluster.
6. If the above items have been checked and this problem persists over several days causing repeats of this error message in the local queue manager's error logs, contact your IBM support center.

AMQ9487

Remote queue manager is a standby queue manager.

Severity

30 : Severe error

Explanation

Channel <*insert_3*> is closing because the remote queue manager is a standby queue manager.

Response

None.

AMQ9488

Program cannot connect to the standby queue manager.

Severity

30 : Severe error

Explanation

The connection attempt to queue manager <*insert_4*> failed with reason code <*insert_1*> because the queue manager is a standby queue manager.

Response

Standby queue managers do not accept connections. Connect to the primary queue manager instead.

AMQ9489

The maximum number of instances, <*insert_1*> , of channel <*insert_3*> was reached.

Severity

30 : Severe error

Explanation

The server-connection channel <*insert_3*> is configured so that the maximum number of instances that can run at the same time is <*insert_1*>. This limit was reached.

Response

Try the operation again when a new instance can be started.

If the limit has been reached because there are too many connections from one or more of your client applications, consider changing the applications to make fewer connections.

If you are not making use of sharing conversations, consider switching to this mode of operation because several client connections can then share one channel instance.

AMQ9490

The maximum number of instances, <*insert_1*> , of channel <*insert_3*> was reached for an individual client.

Severity

30 : Severe error

Explanation

The server-connection channel <*insert_3*> is configured so that the maximum number of instances that can run at the same time for any individual client is <*insert_1*> . This limit was reached for the client with remote network address <*insert_4*>.

Response

Try the operation again when a new instance can be started for this client.

If the limit has been reached because there are too many connections from the relevant client application, consider changing the application to make fewer connections.

If you are not making use of sharing conversations, consider switching to this mode of operation because several client connections can then share one channel instance.

AMQ9491

Transmission Queue <*insert_3*> set to NOSHARE.

Severity

20 : Error

Explanation

The channel <*insert_4*> on queue manager <*insert_5*> cannot start because this queue manager has a setting for PipeLineLength greater than 1, and so multiple threads will run in this channel's MCA. Only the first thread would be able to open the Transmission Queue <*insert_3*> because it is set to be non-shareable.

Response

Check the definition of the Transmission Queue <*insert_3*> on queue manager <*insert_5*> and set it to be SHARE instead of NOSHARE. Alternatively, you can set all channels on this queue manager to use only a single thread, by using the PipeLineLength parameter.

AMQ9492

The <*insert_3*> responder program encountered an error.

Severity

30 : Severe error

Explanation

The responder program was started but detected an error.

Response

Look at previous error messages in the error files to determine the error encountered by the responder program.

AMQ9494

A protocol error was detected for channel <*insert_3*> .

Severity

30 : Severe error

Explanation

During communications with the remote queue manager, a TCP/IP read and receive call returned EINTR, indicating that it had been interrupted. Immediately after this the channel program detected a protocol error. The failure type was <*insert_1*> with associated data of <*insert_2*>.

Response

If you are running an AIX client you will avoid problems arising from EINTRs on TCP/IP reads, by writing your application so that system calls interrupted by signals are restarted. You must establish the signal handler with sigaction(2) and set the SA_RESTART flag in the sa_flags field of the new action structure. If you are running on a platform other than AIX, an AIX server, or an AIX client with an application that adheres to the restart guidelines provided above, contact the systems administrator who should examine the error logs to determine the cause of the failure.

AMQ9495

The CLWL exit <*insert_3*> is inconsistent with a dynamic cache.

Severity

30 : Severe error

Explanation

When the CLWL exit <*insert_3*> was called for the ExitReason MQXR_INIT, the value <*insert_1*> was returned in the ExitResponse2 field. This indicates the CLWL exit is incompatible with the Queue Manager cache type which is dynamic. Either change the Queue Manager cache type to static (using the Tuning Parameter, ClusterCacheType=STATIC) or rewrite the CLWL exit to be compatible with a dynamic cache". The CLWL exit has been suppressed.

Response

None.

AMQ9496

Channel ended by a remote exit.

Severity

30 : Severe error

Explanation

Channel program <*insert_3*> was ended because the channel exit at the remote end requested it.

Response

Examine the error logs at the remote end of the channel to see the reason why the remote exit ended the channel.

AMQ9498

The MQCD structure supplied was not valid.

Severity

30 : Severe error

Explanation

The value of the <*insert_3*> field has the value <*insert_4*>. This value is invalid for the operation requested.

Response

Change the parameter and retry the operation.

AMQ9499

A WebSphere MQ listener will end shortly.

Severity

0 : Information

Explanation

One listener detected in the system is scheduled for shutdown.

Response

None.

AMQ9500

No Repository storage

Severity
10 : Warning

Explanation
An operation failed because there was no storage available in the repository. An attempt was made to allocate <*insert_1*> bytes from <*insert_3*>.

Response
Reconfigure the Queue Manager to allocate a larger repository.

AMQ9501
Usage: <*insert_3*> [-m QMgrName] -c ChlName.

Severity
10 : Warning

Explanation
Values passed to the channel program are not valid. The parameter string passed to this program is as follows :- [-m QMgrName] -c ChlName Default values will be used for parameters not supplied.

Response
Correct the parameters passed to the Channel program and retry the operation.

AMQ9502
Type of channel not suitable for action requested.

Severity
30 : Severe error

Explanation
The operation requested cannot be performed on channel <*insert_3*>. Some operations are only valid for certain channel types. For example, you can only ping a channel from the end sending the message.

Response
Check whether the channel name is specified correctly. If it is check that the channel has been defined correctly.

AMQ9503
Channel negotiation failed.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> between this machine and the remote machine could not be established due to a negotiation failure.

Response
Tell the systems administrator, who should attempt to identify the cause of the channel failure using problem determination techniques. For example, look for FFST files, and examine the error logs on the local and remote systems where there may be messages explaining the cause of failure. More information may be obtained by repeating the operation with tracing enabled.

AMQ9504
A protocol error was detected for channel <*insert_3*> .

Severity
30 : Severe error

Explanation
During communications with the remote queue manager, the channel program detected a protocol error. The failure type was <*insert_1*> with associated data of <*insert_2*>.

Response
Contact the systems administrator who should examine the error logs to determine the cause of the failure.

AMQ9505
Channel sequence number wrap values are different.

Severity
30 : Severe error

Explanation
The sequence number wrap value for channel <*insert_3*> is <*insert_1*>, but the value specified at the remote location is <*insert_2*>. The two values must be the same before the channel can be started.

Response
Change either the local or remote channel definitions so that the values specified for the message sequence number wrap values are the same.

AMQ9506
Message receipt confirmation failed.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> has ended because the remote queue manager did not accept the last batch of messages.

Response
The error log for the channel at the remote site will contain an explanation of the failure. Contact the remote Systems Administrator to resolve the problem.

AMQ9507
Channel <*insert_3*> is currently in-doubt.

Severity
30 : Severe error

Explanation
The requested operation cannot complete because the channel is in-doubt with host <*insert_4*>.

Response
Examine the status of the channel, and either restart a channel to resolve the in-doubt state, or use the RESOLVE CHANNEL command to correct the problem manually.

AMQ9508
Program cannot connect to the queue manager.

Severity
30 : Severe error

Explanation
The connection attempt to queue manager <*insert_4*> failed with reason code <*insert_1*>.

Response
Ensure that the queue manager is available and operational.

AMQ9509
Program cannot open queue manager object.

Severity
30 : Severe error

Explanation
The attempt to open either the queue or queue manager object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response
Ensure that the queue is available and retry the operation.

AMQ9510
Messages cannot be retrieved from a queue.

Severity
30 : Severe error

Explanation
The attempt to get messages from queue <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response
If the reason code indicates a conversion problem, for example, MQRC_SOURCE_CCSID_ERROR, remove the message(s) from the queue. Otherwise, ensure that the required queue is available and operational.

AMQ9511
Messages cannot be put to a queue.

Severity
30 : Severe error

Explanation
The attempt to put messages to queue <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response
Ensure that the required queue is available and operational.

AMQ9512
Ping operation is not valid for channel <*insert_3*> .

Severity
30 : Severe error

Explanation
Ping may only be issued for SENDER, SERVER or CLUSSDR channel types. Also, it may not be issued for an SSL channel on the HP-UX or Linux platforms.

Response
If the local channel is a receiver channel, you must issue the ping from the remote queue manager.

AMQ9513
Maximum number of channels reached.

Severity
30 : Severe error

Explanation
The maximum number of channels that can be in use simultaneously has been reached. The number of permitted channels is a configurable parameter in the queue manager configuration file.

Response
Wait for some of the operating channels to close. Retry the operation when some channels are available.

AMQ9514
Channel <*insert_3*> is in use.

Severity
30 : Severe error

Explanation
The requested operation failed because channel <*insert_3*> is currently active.

Response
Either end the channel manually, or wait for it to close, and retry the operation.

AMQ9515
Channel <*insert_3*> changed.

Severity
10 : Warning

Explanation
The statistics shown are for the channel requested, but it is a new instance of the channel. The previous channel instance has ended.

Response
None.

AMQ9516
File error occurred.

Severity
30 : Severe error

Explanation
The filesystem returned error code <*insert_1*> for file <*insert_3*>.

Response
Record the name of the file <*insert_3*> and tell the systems administrator, who should ensure that file <*insert_3*> is correct and available.

AMQ9516 (IBM i)
File error occurred.

Severity
30 : Severe error

Explanation
The filesystem returned error code <*insert_4*> for file <*insert_3*>.

Response
Record the name of the file <*insert_3*> and tell the systems administrator, who should ensure that file <*insert_3*> is correct and available.

AMQ9517
File damaged.

Severity
30 : Severe error

Explanation
The program has detected damage to the contents of file <*insert_3*>.

Response
Record the values and tell the systems administrator who must restore a saved version of file <*insert_3*>. The return code was <*insert_1*> and the record length returned was <*insert_2*>.

AMQ9518
File <*insert_3*> not found.

Severity
30 : Severe error

Explanation
The program requires that the file <*insert_3*> is present and available.

Response
This may be caused by invalid values for the optional environment variables MQCHLLIB, MQCHLTAB or MQDATA. If these variables are valid or not set then record the name of the file and tell the systems administrator who must ensure that file <*insert_3*> is available to the program.

AMQ9519
Channel <*insert_3*> not found.

Severity
30 : Severe error

Explanation
The requested operation failed because the program could not find a definition of channel <*insert_3*>.

Response
Check that the name is specified correctly and the channel definition is available.

AMQ9520
Channel not defined remotely.

Severity
30 : Severe error

Explanation
There is no definition of channel <*insert_3*> at the remote location.

Response
Add an appropriate definition to the remote hosts list of defined channels and retry the operation.

AMQ9521
Host is not supported by this channel.

Severity
30 : Severe error

Explanation
The connection across channel <*insert_5*> was refused because the remote host <*insert_4*> did not match the host <*insert_3*> specified in the channel definition.

Response
Update the channel definition, or remove the explicit mention of the remote machine connection name.

AMQ9522
Error accessing the status table.

Severity
30 : Severe error

Explanation
The program could not access the channel status table.

Response
A value of <*insert_1*> was returned from the subsystem when an attempt was made to access the Channel status table. Contact the systems administrator, who should examine the log files to determine why the program was unable to access the status table.

AMQ9523
Remote host detected a protocol error.

Severity
30 : Severe error

Explanation
During communications through channel <*insert_3*> , the remote queue manager channel program detected a protocol error. The failure type was <*insert_1*> with associated data of <*insert_2*>.

Response
Tell the systems administrator, who should examine the error files to determine the cause of the failure.

AMQ9524
Remote queue manager unavailable.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> cannot start because the remote queue manager is not currently available.

Response
Either start the remote queue manager, or retry the operation later.

AMQ9525
Remote queue manager is ending.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> is closing because the remote queue manager is ending.

Response
None.

AMQ9526
Message sequence number error for channel <*insert_3*> .

Severity
30 : Severe error

Explanation
The local and remote queue managers do not agree on the next message sequence number. A message with sequence number <*insert_1*> has been sent when sequence number <*insert_2*> was expected. The remote host is <*insert_4*>.

Response
Determine the cause of the inconsistency. It could be that the synchronization information has become damaged, or has been backed out to a previous version. If the situation cannot be resolved, the sequence number can be manually reset at the sending end of the channel using the RESET CHANNEL command.

AMQ9527
Cannot send message through channel <*insert_3*> .

Severity
30 : Severe error

Explanation
The channel has closed because the remote queue manager cannot receive a message.

Response
Contact the systems administrator who should examine the error files of the remote queue manager, to determine why the message cannot be received, and then restart the channel.

AMQ9528
User requested closure of channel <*insert_3*> .

Severity
10 : Warning

Explanation
The channel is closing because of a request by the user.

Response
None.

AMQ9529
Target queue unknown on remote host.

Severity
30 : Severe error

Explanation
Communication using channel <*insert_3*> has ended because the target queue for a message is unknown at the remote host.

Response
Ensure that the remote host contains a correctly defined target queue, and restart the channel.

AMQ9530
Program could not inquire queue attributes.

Severity
30 : Severe error

Explanation
The attempt to inquire the attributes of queue <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response
Ensure that the queue is available and retry the operation.

AMQ9531
Transmission queue specification error.

Severity
30 : Severe error

Explanation
Queue <*insert_4*> identified as a transmission queue in the channel definition <*insert_3*> is not a transmission queue.

Response
Ensure that the queue name is specified correctly. If so, alter the queue usage parameter of the queue to that of a transmission queue.

AMQ9532
Program cannot set queue attributes.

Severity
30 : Severe error

Explanation
The attempt to set the attributes of queue <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response
Ensure that the queue is available and retry the operation.

AMQ9533
Channel <*insert_3*> is not currently active.

Severity
10 : Warning

Explanation
The channel was not stopped because it was not currently active. If attempting to stop a specific instance of a channel by connection name or by remote queue manager name this message indicates that the specified instance of the channel is not running.

Response
None.

AMQ9534
Channel <*insert_3*> is currently not enabled.

Severity
30 : Severe error

Explanation
The channel program ended because the channel is currently not enabled.

Response
Issue the START CHANNEL command to re-enable the channel.

AMQ9535
User exit not valid.

Severity
30 : Severe error

Explanation
Channel program <*insert_3*> ended because user exit <*insert_4*> is not valid.

Response
Ensure that the user exit is specified correctly in the channel definition, and that the user exit program is correct and available.

AMQ9536
Channel ended by an exit.

Severity
30 : Severe error

Explanation
Channel program <*insert_3*> was ended by exit <*insert_4*>.

Response
None.

AMQ9537
Usage: <*insert_3*> [-m QMgrName] [-q InitQ]

Severity
10 : Warning

Explanation
Values passed to the Channel Initiator program are not valid. The parameters should be passed as follows: [-m QMgrName] [-q InitQ] Default values are used for parameters that are not supplied.

Response
Correct the parameters passed to the program and retry the operation.

AMQ9538
Commit control error.

Severity
30 : Severe error

Explanation
An error occurred when attempting to start commitment control. Either exception <*insert_3*> was received when querying commitment status, or commitment control could not be started.

Response
Refer to the error log for other messages pertaining to this problem.

AMQ9539
No channels available.

Severity
30 : Severe error

Explanation
The channel initiator program received a trigger message to start an MCA program to process queue <*insert_3*>. The program could not find a defined, available channel to start.

Response
Ensure that there is a defined channel, which is enabled, to process the transmission queue.

AMQ9540
Commit failed.

Severity
30 : Severe error

Explanation
The program ended because return code <*insert_1*> was received when an attempt was made to commit change to the resource managers. The commit ID was <*insert_3*>.

Response
Tell the systems administrator.

AMQ9541
CCSID supplied for data conversion not supported.

Severity
30 : Severe error

Explanation
The program ended because, either the source CCSID <*insert_1*> or the target CCSID <*insert_2*> is not valid, or is not currently supported.

Response
Correct the CCSID that is not valid, or ensure that the requested CCSID can be supported.

AMQ9542
Queue manager is ending.

Severity
10 : Warning

Explanation
The program will end because the queue manager is quiescing.

Response
None.

AMQ9543
Status table damaged.

Severity
30 : Severe error

Explanation
The channel status table has been damaged.

Response
End all running channels and issue a DISPLAY CHSTATUS command to see the status of the channels. Use the standard facilities supplied with your system to record the problem identifier and to save any generated output files. Use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9544
Messages not put to destination queue.

Severity
10 : Warning

Explanation
During the processing of channel <*insert_3*> one or more messages could not be put to the destination queue and attempts were made to put them to a dead-letter queue. The location of the queue is <*insert_1*>, where 1 is the local dead-letter queue and 2 is the remote dead-letter queue.

Response
Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed. Also look at previous error messages to see if the attempt to put messages to a dead-letter queue failed. The program identifier (PID) of the processing program was <*insert_4*>.

AMQ9545
Disconnect interval expired.

Severity
0 : Information

Explanation
Channel <*insert_3*> closed because no messages arrived on the transmission queue within the disconnect interval period.

Response
None.

AMQ9546
Error return code received.

Severity
30 : Severe error

Explanation
The program has ended because return code <*insert_1*> was returned from function <*insert_3*>

Response
Correct the cause of the failure and retry the operation.

AMQ9547
Type of remote channel not suitable for action requested.

Severity
30 : Severe error

Explanation
The operation requested cannot be performed because channel <*insert_3*> on the remote machine is not of a suitable type. For example, if the local channel is defined as a sender the remote machine must define its channel as either a receiver or requester.

Response
Check that the channel name is specified correctly. If it is, check that the remote channel has been defined correctly.

AMQ9548
Message put to the 'dead-letter queue'.

Severity
10 : Warning

Explanation
During processing a message has been put to the dead-letter queue.

Response
Examine the contents of the dead-letter queue. Each message is contained in a structure that describes why the message was put to the queue, and to where it was originally addressed.

AMQ9549
Transmission Queue <*insert_3*> inhibited for MQGET.

Severity
20 : Error

Explanation
An MQGET failed because the transmission queue had been previously inhibited for MQGET.

Response
None.

AMQ9550
Channel program <*insert_3*> cannot be stopped at this time.

Severity
30 : Severe error

Explanation
The channel program cannot be terminated immediately but should end shortly.

Response
If the channel does not end in a short time issue the STOP CHANNEL command again.

AMQ9551
Protocol not supported by remote host

Severity
30 : Severe error

Explanation
The operation you are performing over Channel <*insert_3*> to the host at <*insert_4*> is not supported by the target host.

Response
Check that the connection name parameter is specified correctly and that the levels of the products in use are compatible.

AMQ9552
Security flow not received.

Severity
30 : Severe error

Explanation
During communications through channel <*insert_3*> the local security exit requested security data from the remote machine. The security data has not been received so the channel has been closed.

Response
Tell the systems administrator who should ensure that the security exit on the remote machine is defined correctly.

AMQ9553
The function is not supported.

Severity
30 : Severe error

Explanation
The <*insert_3*> function <*insert_4*> attempted is not currently supported on this platform.

Response
None.

AMQ9554
User not authorized.

Severity
30 : Severe error

Explanation
You are not authorized to perform the Channel operation.

Response
Tell the systems administrator who should ensure that the correct access permissions are available to you, and then retry the operation.

AMQ9555
File format error.

Severity
30 : Severe error

Explanation
The file <*insert_3*> does not have the expected format.

Response
Ensure that the file name is specified correctly.

AMQ9556
Channel synchronization file missing or damaged.

Severity
30 : Severe error

Explanation
The channel synchronization file <*insert_3*> is missing or does not correspond to the stored channel information for queue manager <*insert_4*>.

Response
Rebuild the synchronization file using the rcrmqobj command

```
rcrmqobj -t syncfile (-m q-mgr-name)
```

AMQ9556 (IBM i)
Channel synchronization file missing or damaged.

Severity
30 : Severe error

Explanation
The channel synchronization file <*insert_3*> is missing or does not correspond to the stored channel information for queue manager <*insert_4*>.

Response
Rebuild the synchronization file using the RCRMQMOBJ command.

AMQ9557
Queue Manager User ID initialization failed.

Severity
30 : Severe error

Explanation
The call to initialize the User ID failed with CompCode <*insert_1*> and Reason <*insert_2*> .

Response
Correct the error and try again.

AMQ9558
The remote channel <*insert_3*> is not currently available.

Severity
30 : Severe error

Explanation
The channel program ended because an instance of channel <*insert_3*> could not be started on the remote system. This could be for one of the following reasons:

The channel is disabled.

The remote system does not have sufficient resources to run another instance of the channel.

In the case of a client-connection channel, the limit on the number of instances configured for the remote server-connection channel was reached.

Response
Check the remote system to ensure that the channel is able to run. Try the operation again.

AMQ9560
Rebuild Synchronization File - program started

Severity
0 : Information

Explanation
Rebuilding the Synchronization file for Queue Manager <*insert_3*> .

Response
None.

AMQ9561
Rebuild Synchronization File - program completed normally

Severity
0 : Information

Explanation
Rebuild Synchronization File program completed normally.

Response
None.

AMQ9562
Synchronization file in use.

Severity
30 : Severe error

Explanation
The Synchronization file <*insert_3*> is in use and cannot be re-created.

Response
Stop any channel activity and retry the rcrmqobj command.

AMQ9562 (IBM i)
Synchronization file in use.

Severity
30 : Severe error

Explanation
The Synchronization file <*insert_3*> is in use and cannot be re-created.

Response
Stop any channel activity and retry the RCRMQMOBJ command.

AMQ9563
Synchronization file cannot be deleted

Severity
30 : Severe error

Explanation
The filesystem returned error code <*insert_1*> for file <*insert_3*>.

Response
Tell the systems administrator who should ensure that file <*insert_3*> is available and not in use.

AMQ9564
Synchronization File cannot be created

Severity
30 : Severe error

Explanation
The filesystem returned error code <*insert_1*> for file <*insert_3*>.

Response
Tell the systems administrator.

AMQ9565
No dead-letter queue defined.

Severity
30 : Severe error

Explanation
The queue manager <*insert_4*> does not have a defined dead-letter queue. A message cannot be transferred across channel <*insert_5*>. The reason code is <*insert_1*>. The destination queue is <*insert_3*> .

Response
Either correct the problem that caused the program to try and write a message to the dead-letter queue or create a dead-letter queue for the queue manager.

AMQ9566
Invalid MQSERVER value

Severity
30 : Severe error

Explanation
The value of the MQSERVER environment variable was <*insert_3*> . The variable should be in the format 'ChannelName/Protocol/ConnectionName'.

Response
Correct the MQSERVER value and retry the operation.

AMQ9572
Message header is not valid.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> is stopping because a message header is not valid. During the processing of the channel, a message was found that has a header that is not valid. The dead-letter queue has been defined as a transmission queue, so a loop would be created if the message had been put there.

Response
Correct the problem that caused the message to have a header that is not valid.

AMQ9573
Maximum number of active channels reached.

Severity
30 : Severe error

Explanation
There are too many channels active to start another. The current defined maximum number of active channels is <*insert_1*> .

Response
Either wait for some of the operating channels to close or use the stop channel command to close some channels. Retry the operation when some channels are available. The maximum number of active channels is a configurable parameter in the queue manager configuration file.

AMQ9574
Channel <*insert_3*> can now be started.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> has been waiting to start, but there were no channels available because the maximum number of active channels was running. One, or more, of the active channels has now closed so this channel can start.

AMQ9575
DCE Security: failed to get the user's login name.

Severity
30 : Severe error

Explanation
System call <*insert_4*> to get the login name of the user running WebSphere MQ MQI client application process <*insert_1*> failed with error value <*insert_2*> . This occurred in security exit function create_cred. The exit will now attempt to open channel <*insert_3*> using the DCE default login context.

Response

If you wish to run using the DCE default login context take no action. If you wish to run using the user's login name as the DCE security exit principal examine the documentation for the operating system on which you are running MQ MQI clients and reconfigure the operating system as necessary to allow the <*insert_4*> call to succeed.

AMQ9576

DCE Security: an exit could not allocate memory.

Severity

30 : Severe error

Explanation

A DCE exit was unsuccessful in obtaining the memory it needed. The failure occurred in exit function <*insert_4*> . Channel <*insert_3*> is closed.

Response

Make more memory available to the WebSphere MQ system and restart the relevant channel.

AMQ9577

DCE security exit: no partner name.

Severity

30 : Severe error

Explanation

Channel <*insert_3*> has not been opened because the DCE security exit which initiates the security context was not passed a valid partner name. When the DCE security exit is called to initiate the security context it is essential that the PartnerName field in the MQCXP structure contains a valid partner name. On this call it did not. This can arise as a result of a usage error, for example, only specifying the security exit on one end of the channel. The error was reported from security exit function savePartnerName.

Response

Check your usage of the DCE security exit for errors, such as only specifying the exit in one of the matching channel definitions. Correct any errors found and retry.

AMQ9578

DCE Security: bad return from DCE call.

Severity

30 : Severe error

Explanation

Channel <*insert_3*> has been closed because one of the DCE channel exits received a bad return code from DCE.

Response

Consult the appropriate DCE manuals to find out the meaning of major_status <*insert_1*> and minor_status <*insert_2*> on call <*insert_5*>. Then rectify the error. The exit function name is <*insert_4*> .

AMQ9579

DCE Security: partner name does not match target.

Severity

30 : Severe error

Explanation

The DCE Security exit was requested to perform a trusted channel check: target partner name <*insert_4*> was specified in the SCYDATA field of channel <*insert_3*>. The actual partner name associated with channel <*insert_3*> was <*insert_5*>, so the security exit suppressed the channel.

Response

Examine the channel definition of channel <*insert_3*> and alter it so that the relevant name on the partner system matches that specified in the SCYDATA field.

AMQ9580

DCE Security: invalid message received.

Severity

30 : Severe error

Explanation

An IBM-supplied DCE exit on channel <*insert_3*> received a message that was not generated by a matching exit, or was not the expected type of message. The header.mechanism field had value <*insert_1*>. The header.msgtype field had value <*insert_2*>. The name of the exit function in which the error was discovered is <*insert_4*> .

Response

Make sure that the exits at both ends of the channel generate compatible flows.

AMQ9581

DCE Security: wrong exit called.

Severity

30 : Severe error

Explanation

Exit <*insert_4*> on channel <*insert_3*> was called for use as a WebSphere MQ exit of the wrong type. DCE_SEC_SCY_CHANNELEXIT functions as a security exit;
DCE_SEC_SRM_CHANNELEXIT functions as a send, receive or message exit. The ExitId parameter passed to the exit was <*insert_1*>.

Response

Alter the exit definitions to ensure that exit <*insert_4*> is called correctly.

AMQ9582

DCE Security: invalid exit function requested.

Severity

30 : Severe error

Explanation

Exit <*insert_4*> on channel <*insert_3*> was called with an invalid ExitReason (value <*insert_1*>).

Response

Check that the exit is being run with a compatible release of WebSphere MQ base code. If not then correct it. If it is, save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9583

The DCE security exit was not run.

Severity

30 : Severe error

Explanation

The DCE_SEC_SRM_CHANNELEXIT exit was called on channel <*insert_3*>; the value of pContext->mechanism (<*insert_1*>) passed was not valid.

Response

This is probably because the DCE_SEC_SRM_CHANNELEXIT exit has been called without first calling the DCE_SEC_SCY_CHANNELEXIT security exit. Alter the system so that either both or neither are run.

AMQ9584
DCE Security: message too short.

Severity
30 : Severe error

Explanation
The DCE_SEC_SRM_CHANNELEXIT receive or message exit was called on channel <*insert_3*> to process an incoming message. The pDataLength parameter supplied to the exit indicated that the message received was too short to be a valid message for the relevant exit. The *pDataLength value was <*insert_1*> .

Response
Configure the system so that compatible send/receive/message exits are run at both ends of the channel.

AMQ9585
Maximum number of channel initiators reached.

Severity
30 : Severe error

Explanation
The maximum number of channels initiators that can be in use simultaneously has been reached. The number of permitted channel initiators is a configurable parameter in the queue manager configuration file.

Response
Wait for one or more channel initiators to close and retry the operation or modify the configuration file to allow more initiators and restart the Queue Manager.

AMQ9586
Program cannot create queue manager object.

Severity
30 : Severe error

Explanation
The attempt to create object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response
Use the standard facilities supplied with your system to record the problem identifier. Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9587
Program cannot open queue manager object.

Severity
30 : Severe error

Explanation
The attempt to open object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response
Use the standard facilities supplied with your system to record the problem identifier. Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at

http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9588

Program cannot update queue manager object.

Severity

30 : Severe error

Explanation

The attempt to update object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier. Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9589

Program cannot query queue manager object.

Severity

30 : Severe error

Explanation

The attempt to query object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier. Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9590

Program cannot close queue manager object.

Severity

30 : Severe error

Explanation

The attempt to close object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier. Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9591

Program cannot prepare queue manager object.

Severity

30 : Severe error

Explanation

The attempt to prepare object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier. Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9592

Program cannot resolve queue manager object.

Severity

30 : Severe error

Explanation

The attempt to resolve object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier. Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9593

Program cannot delete queue manager object.

Severity

30 : Severe error

Explanation

The attempt to delete object <*insert_4*> on queue manager <*insert_5*> failed with reason code <*insert_1*>.

Response

Use the standard facilities supplied with your system to record the problem identifier. Save any generated output files and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9594

Usage: runmqfmt [filename].

Severity

0 : Information

Explanation

Syntax for the usage of runmqfmt.

Response

None.

AMQ9595

Usage: endmqlsr [-w] [-m QMgrName]

Severity

10 : Warning

Explanation

The correct usage is shown.

Response

Correct the parameters passed to the endmqlsr program and retry the operation.

AMQ9596
Queue Manager <*insert_3*> still running

Severity
30 : Severe error

Explanation
The requested operation cannot complete because queue manager <*insert_3*> is still running.

Response
End the queue manager and retry the operation.

AMQ9597
No WebSphere MQ listeners for Queue Manager <*insert_3*> .

Severity
0 : Information

Explanation
No listener processes were found in the system for Queue Manager <*insert_3*>.

Response
None.

AMQ9598
<*insert_1*> WebSphere MQ listeners will end shortly.

Severity
0 : Information

Explanation
<*insert_1*> listeners detected in the system are scheduled for shutdown.

Response
None.

AMQ9599
Program could not open a queue manager object.

Severity
30 : Severe error

Explanation
The attempt to open either the queue or queue manager object <*insert_4*> on queue manager <*insert_5*> by user <*insert_3*> failed with reason code <*insert_1*>.

Response
Ensure that the queue is available and retry the operation. If the message is from a remote Queue Manager, check the Message Channel Agent User Identifier has the correct authority.

AMQ9601
Program could not inquire on queues on this queue manager.

Severity
30 : Severe error

Explanation
The WebSphere MQ clustering repository program was attempting to find out about the queues on queue manager <*insert_3*> . One of the calls failed with reason code <*insert_1*> . The repository command was backed out and the repository process went into a timed wait.

Response
Correct the error. When the repository process restarts it processes the backed out command again and continues.

AMQ9602
Maximum number of channel processes reached.

Severity
30 : Severe error

Explanation
The channel cannot start because the number of channel processes has already reached the maximum allowable value. The maximum number of channels processes is configured as <*insert_1*> . This value is a configurable parameter in the queue manager configuration file.

Response
Wait for some of the operating channels to close. Retry the operation when some channels are available.

AMQ9603
Error accessing the process pool shared segment.

Severity
30 : Severe error

Explanation
The program could not access the process pool shared segment

Response
A value of <*insert_1*> was returned from the subsystem when an attempt was made to access the Channel process pool shared memory. Contact the systems administrator, who should examine the log files to determine why the program was unable to access the process pool shared segment.

AMQ9604
Channel <*insert_3*> terminated unexpectedly

Severity
30 : Severe error

Explanation
The process or thread executing channel <*insert_3*> is no longer running. The check process system call returned <*insert_1*> for process <*insert_2*> .

Response
No immediate action is required because the channel entry has been removed from the list of running channels. Inform the system administrator who should examine the operating system procedures to determine why the channel process has terminated.

AMQ9605
<*insert_1*> WebSphere MQ listeners have been ended.

Severity
0 : Information

Explanation
<*insert_1*> listeners detected in the system have been ended.

Response
None.

AMQ9606
A WebSphere MQ listener has ended.

Severity
0 : Information

Explanation
One listener detected in the system has been ended.

Response
None.

AMQ9608
Remote resources in recovery

Severity
30 : Severe error

Explanation
Channel <*insert_3*> could not establish a successful connection with the remote Queue Manager because resources are being recovered.

Response
Restart the channel at a later time. If the problem persists then examine the error logs of the remote Queue Manager to see the full explanation of the cause of the problem.

AMQ9610
AMQ<*insert_1*> messages suppressed

Severity
0 : Information

Explanation
<*insert_2*> messages of type AMQ <*insert_1*> were suppressed

Response
Message suppression is controlled by MQ_CHANNEL_SUPPRESS_MSGS and MQ_CHANNEL_SUPPRESS_INTERVAL environment variables.

AMQ9611
Rebuild Client Channel Table - program completed normally

Severity
0 : Information

Explanation
Rebuild Client Channel Table program completed normally.

Response
None.

AMQ9612
<*insert_1*> WebSphere MQ listeners could not be ended.

Severity
0 : Information

Explanation
The request to the end the WebSphere MQ listeners for specified Queue Manager was completed however <*insert_1*> listeners could not be stopped. Reasons why listener may not be stopped are:

The listener process contains channels which are still active.

Response
Active channels may be stopped using the 'STOP CHANNEL' command or by ending the Queue Manager, and reissuing the end-listener request.

AMQ9614 (IBM i)
Certificate is not signed by a trusted Certificate Authority.

Severity
0 : Information

Explanation
The attempt to start channel <*insert_3*> failed because the certificate used in the SSL handshake is

not signed by a Certificate Authority (CA) listed in the certificate trust list for this queue manager. This error occurs when the SSL key repository for the queue manager is specified as '*SYSTEM' and the application definition in Digital Certificate Manager has been modified to specify a CA trust list.

Response

Use Digital Certificate Manager to add the required Certificate Authority (CA) certificates to the application definitions CA trust list.

AMQ9615 (IBM i)

Queue Manager is not registered with DCM.

Severity

0 : Information

Explanation

The attempt to start channel <*insert_3*> failed because the queue manager is not registered as a SSL server application with Digital Certificate Manager (DCM). This error occurs when the SSL key repository for the queue manager is specified as '*SYSTEM' but WebSphere MQ cannot register the queue manager as an SSL server application with DCM, or alternatively when the application definition for the queue manager has been manually removed from DCM.

Response

Attempt to re-register the queue manager with Digital Certificate Manager by issuing CHGMQM SSLKEYR(*SYSTEM). If this is unsuccessful you may need to manually add the application definition through Digital Certificate Manager, see the WebSphere MQ Security manual for more details.

AMQ9616

The CipherSpec proposed is not enabled on the server.

Severity

30 : Severe error

Explanation

The SSL or TLS subsystem at the server end of a channel been configured in such a way that it has rejected the CipherSpec proposed by an SSL or TLS client. This rejection occurred during the secure socket handshake (i.e. it happened before the proposed CipherSpec was compared with the CipherSpec in the server channel definition).

This error most commonly occurs when the choice of acceptable CipherSpecs has been limited in one of the following ways:

(a) The server queue manager SSLFipsRequired attribute is set to YES and the channel is using a CipherSpec which is not FIPS-certified on the server.

(b) The server queue manager EncyptionPolicySuiteB attribute has been set to a value other than NONE and the channel is using a CipherSpec which does not meet the server's configured Suite B security level.

The channel is '<*insert_3*>', in some cases its name cannot be determined and is shown as '????'. The channel did not start.

Response

Analyse why the proposed CipherSpec was not enabled on the SSL server. Alter the client CipherSpec, or reconfigure the SSL server to accept the original client CipherSpec. Restart the channel.

This message might occur after applying WebSphere MQ maintenance because the FIPS and Suite B standards are updated periodically. When such changes occur, WebSphere MQ is also updated to implement the latest standard. As a result, you might see changes in behavior after applying maintenance. For more information about the versions of FIPS and Suite B standards enforced by WebSphere MQ, see the readme file.

AMQ9617
Parameter requesting FIPS has an invalid value.

Severity
30 : Severe error

Explanation
An SSL channel running on an MQ MQI client has failed to start. This is because the value specified for the MQSSLFIPS environment variable, or in the MQSCO FipsRequired field, is invalid. The value specified was "<*insert_3*>".

Response
Set the MQSSLFIPS environment variable, or the MQSCO FipsRequired field, to a valid value. Restart the channel.

AMQ9618
SSLCRLNL attribute points to a namelist with no names.

Severity
30 : Severe error

Explanation
An SSL channel has failed to start because the SSLCRLNL queue manager attribute points to a namelist with an empty list of names.

Response
If OCSP or CRL checking is required, set up the namelist referenced by SSLCRLNL with a non-empty list of authentication information object names. If no OCSP or CRL checking is required, clear the SSLCRLNL queue manager attribute. Restart the failing channel.

AMQ9619
SSL cannot be run from an unthreaded HP-UX MQ MQI client.

Severity
30 : Severe error

Explanation
On HP-UX, SSL cannot be run from a WebSphere MQ MQI client which was linked with the unthreaded client libraries.

Response
Either relink your client application with the threaded client libraries, or do not attempt to use SSL from this application.

AMQ9620
Internal error on call to SSL function on channel <*insert_3*> .

Severity
30 : Severe error

Explanation
An error indicating a software problem was returned from a function which is used to provide SSL support. The error code returned was <*insert_1*>. The function call was <*insert_4*> . The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9620 (IBM i)

Unexpected SSL error on call to <*insert_4*> .

Severity

0 : Information

Explanation

An unexpected SSL error was returned from function <*insert_4*> for channel <*insert_3*>. The error code returned was <*insert_1*>. GSKit error codes are documented in the MQ manuals and also in the GSKSSL member of the H file in library QSYSINC.

Response

Collect the items listed in the 'Problem determination' section of the System Administration manual and use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center.

AMQ9621

Error on call to SSL function ignored on channel <*insert_3*> .

Severity

10 : Warning

Explanation

An error indicating a software problem was returned from a function which is used to provide SSL support. The error code returned was <*insert_1*>. The function call was <*insert_4*> . The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. This error is not regarded as sufficiently serious to interrupt channel operation; channel operation was not affected.

Response

None.

AMQ9622

AUTHINFO object <*insert_1*> does not exist.

Severity

30 : Severe error

Explanation

A channel or channel process has failed to start because the namelist of AUTHINFO objects includes the name <*insert_1*>, but no AUTHINFO object of that name exists.

Response

Ensure all the names in the namelist specified on the SSLCRLNL queue manager attribute correspond to AUTHINFO objects which are to be used on the SSL channels. Restart the failing channel or channel process.

AMQ9623

Error inquiring on AUTHINFO object <*insert_3*> .

Severity

30 : Severe error

Explanation

A channel or channel process has failed to start because reason code <*insert_1*> was returned when an inquire was performed on AUTHINFO object <*insert_3*>.

Response

Look at the MQRC_ values in the WebSphere MQ Application Programming Reference to determine the meaning of reason code <*insert_1*> , correct the error, and restart the failing channel or channel process.

AMQ9624
AUTHINFO object <*insert_3*> is not of type CRLLDAP or OCSP.

Severity
30 : Severe error

Explanation
A channel or channel process has failed to start because one of the AUTHINFO objects specified in the SSLCRLNL namelist does not have a valid AUTHTYPE. Instead the type value is <*insert_1*> .

Response
Include only AUTHINFO objects with AUTHTYPE CRLLDAP or AUTHTYPE OCSP in the namelist specified on the SSLCRLNL queue manager attribute. Restart the channel or channel process.

AMQ9625
AUTHINFO object <*insert_3*> was specified with an invalid CONNAME.

Severity
30 : Severe error

Explanation
A channel or channel process has failed to start because one of the AUTHINFO objects specified in the SSLCRLNL namelist has an invalid CONNAME parameter. The invalid value is <*insert_4*> .

Response
Correct the invalid parameter. Restart the channel or channel process.

AMQ9626
Channel hanging while initializing SSL.

Severity
30 : Severe error

Explanation
The current channel cannot start because another channel is hanging while initializing the SSL subsystem.

Response
Investigate the reason for the hang on the other channel. Once this is rectified, restart this channel.

AMQ9627
The path and stem name for the SSL key repository have not been specified.

Severity
30 : Severe error

Explanation
The directory path and file stem name for the SSL key repository have not been specified. On a MQ MQI client system there is no default location for this file. SSL connectivity is therefore impossible as this file cannot be accessed.

Response
Use the MQSSLKEYR environment variable or MQCONNX API call to specify the directory path and file stem name for the SSL key repository.

AMQ9628
An LDAP server containing CRLs was specified with an invalid CONNAME.

Severity
30 : Severe error

Explanation

The WebSphere MQ MQI client has failed to connect because an invalid CONNAME was found for one of the LDAP servers containing CRLs. The invalid value is <*insert_3*>.

Response

Correct the invalid parameter. If the LDAP details were defined on a queue manager system, regenerate the client definitions. Reconnect.

AMQ9629

Bad SSL cryptographic hardware parameters.

Severity

30 : Severe error

Explanation

The following string was supplied to specify or control use of SSL cryptographic hardware: <*insert_4*>. This string does not conform to any of the MQ SSL cryptographic parameter formats. The channel is <*insert_3*>. The channel did not start.

Response

Correct your SSL cryptographic hardware parameters and restart the channel.

AMQ9630

An expired SSL certificate was loaded.

Severity

30 : Severe error

Explanation

An SSL certificate that was loaded was not corrupt, but failed validation checks on its date fields. The certificate has either expired, or its date is not valid yet (that is, the from date is later than today), or the validity date range is incorrect (for example, the to date is earlier than the from date).

Response

Ensure that the specified SSL certificate has a valid expiry date.

AMQ9631

The CipherSpec negotiated during the SSL handshake does not match the required CipherSpec for channel <*insert_3*> .

Severity

30 : Severe error

Explanation

There is a mismatch between the CipherSpecs on the local and remote ends of channel <*insert_3*>. The channel will not run until this mismatch is resolved. The CipherSpec required in the local channel definition is <*insert_4*> . The name of the CipherSpec negotiated during the SSL handshake is <*insert_5*>. A code is displayed if the name of the negotiated CipherSpec cannot be determined.

Response

Change the channel definitions for <*insert_3*> so the two ends have matching CipherSpecs and restart the channel. If the certificate in use by one end of the channel is a Global Server Certificate, then the negotiated CipherSpec may not match that specified on either end of the channel. This is because the SSL protocol allows a Global Server Certificate to automatically negotiate a higher level of encryption. In these cases specify a CipherSpec which meets the requirements of the Global Server Certificate.

AMQ9631 (IBM i)

The CipherSpecs at the ends of channel <*insert_3*> do not match.

Severity

30 : Severe error

Explanation

There is a mismatch between the CipherSpecs on the local and remote ends of channel <*insert_3*>. The channel will not run until this mismatch is resolved. The local CipherSpec is <*insert_4*> and the remote CipherSpec is <*insert_5*>.

Response

Change the channel definition for <*insert_3*> so that both ends have matching CipherSpecs and restart the channel.

AMQ9633

Bad SSL certificate for channel <*insert_3*> .

Severity

30 : Severe error

Explanation

A certificate encountered during SSL handshaking is regarded as bad for one of the following reasons:

(a) it was formatted incorrectly and could not be validated

(b) it was formatted correctly but failed validation against the Certificate Authority (CA) root and other certificates held on the local system

(c) it was found in a Certification Revocation List (CRL) on an LDAP server

(d) a CRL was specified but the CRL could not be found on the LDAP server

(e) an OCSP responder has indicated that it is revoked

The channel is <*insert_1*> ; in some cases its name cannot be determined and so is shown as '????'. The remote host is '<*insert_3*>'. The channel did not start.

The details of the certificate which could not be validated are '<*insert_2*>'.

The certificate validation error was 2222.

Response

Check which of the possible causes applies on your system. Correct the error, and restart the channel.

AMQ9634

SSL security context expired.

Severity

30 : Severe error

Explanation

During an SSL operation to encrypt or decrypt a secured message, the SSL security context, which is used to secure communications and was previously established with the remote party, has expired because the remote party has shut down. The secured message has not been encrypted or decrypted. This failure has closed WebSphere MQ channel name <*insert_3*>. If the name is '????', the name is unknown. The SSL operation was <*insert_4*> and its completion code was <*insert_5*>.

Response

Determine why the remote party has shut down and if necessary re-start the channel. The shut down might be the result of controlled termination by a system administrator, or the result of an unexpected termination due to an error. The SSL operation is described in the Windows Schannel reference manual.

AMQ9635
Channel <*insert_3*> did not specify a valid CipherSpec.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> did not specify a valid CipherSpec.

Response
Change channel <*insert_3*> to specify a valid CipherSpec.

AMQ9635 (IBM i)
Channel <*insert_3*> did not specify a valid CipherSpec.

Severity
30 : Severe error

Explanation
Channel <*insert_3*> did not specify a valid CipherSpec, or it specified a CipherSpec that is not available from the IBM Cryptographic Access Provider product installed on this machine. CipherSpecs that use 128-bit encryption algorithms are only available in 5722-AC3 (128-bit) IBM Cryptographic Access Provider.

Response
Change channel <*insert_3*> to specify a valid CipherSpec that is available from the IBM Cryptographic Access Provider product installed on this machine. Check that the CipherSpec you are using is available on this machine in either the 5722-AC2 (56-bit) IBM Cryptographic Access Provider or 5722-AC3 (128-bit) IBM Cryptographic Access Provider licensed program.

AMQ9636
SSL distinguished name does not match peer name, channel <*insert_3*>.

Severity
30 : Severe error

Explanation
The distinguished name, <*insert_4*>, contained in the SSL certificate for the remote end of the channel does not match the local SSL peer name for channel <*insert_3*> . The distinguished name at the remote end must match the peer name specified (which can be generic) before the channel can be started.

Response
If this remote system should be allowed to connect, either change the SSL peer name specification for the local channel so that it matches the distinguished name in the SSL certificate for the remote end of the channel, or obtain the correct certificate for the remote end of the channel. Restart the channel.

AMQ9637
Channel is lacking a certificate.

Severity
30 : Severe error

Explanation
The channel is lacking a certificate to use for the SSL handshake. The channel name is <*insert_3*> (if '????' it is unknown at this stage in the SSL processing). The channel did not start.

Response
Make sure the appropriate certificates are correctly configured in the key repositories for both ends of the channel.

If you have migrated from WebSphere MQ V5.3 to V6, it is possible that the missing certificate is due to a failure during SSL key repository migration. Check the relevant error logs. If these show

that an orphan certificate was encountered then you should obtain the relevant missing certificate authority (signer) certificates and then import these and the orphan certificate into the WebSphere MQ V6 key repository, and then re-start the channel.

AMQ9638
SSL communications error for channel <*insert_3*> .

Severity
30 : Severe error

Explanation
An unexpected SSL communications error occurred for a channel, as reported in the preceding messages. The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Investigate the problem reported in the preceding messages. Review the local and remote console logs for reports of network errors. Correct the errors and restart the channel.

AMQ9639
Remote channel <*insert_3*> did not specify a CipherSpec.

Severity
30 : Severe error

Explanation
Remote channel <*insert_3*> did not specify a CipherSpec when the local channel expected one to be specified. The channel did not start.

Response
Change the remote channel <*insert_3*> to specify a CipherSpec so that both ends of the channel have matching CipherSpecs.

AMQ9640
SSL invalid peer name, channel <*insert_3*> , attribute <*insert_5*>.

Severity
30 : Severe error

Explanation
The SSL peer name for channel <*insert_3*> includes a distinguished name attribute key <*insert_5*> which is invalid or unsupported. The channel did not start.

Response
Correct the SSL peer name for the channel. Restart the channel.

AMQ9641
Remote CipherSpec error for channel <*insert_3*> .

Severity
30 : Severe error

Explanation
The remote end of channel <*insert_3*> has had a CipherSpec error. The channel did not start.

Response
Review the error logs on the remote system to discover the problem with the CipherSpec.

AMQ9642
No SSL certificate for channel <*insert_3*> .

Severity
30 : Severe error

Explanation

The channel <*insert_3*> did not supply a certificate to use during SSL handshaking, but a certificate is required by the remote queue manager. The channel did not start.

Response

Ensure that the key repository of the local queue manager or MQ MQI client contains an SSL certificate which is associated with the queue manager or client. Alternatively, if appropriate, change the remote channel definition so that its SSLCAUTH attribute is set to OPTIONAL and it has no SSLPEER value set.

If you have migrated from WebSphere MQ V5.3 to V6, it is possible that the missing certificate is due to a failure during SSL key repository migration. Check the relevant error logs. If these show that an orphan certificate was encountered then you should obtain the relevant missing certificate authority (signer) certificates and then import these and the orphan certificate into the WebSphere MQ V6 key repository, and then re-start the channel.

AMQ9642 (IBM i)

No SSL certificate for channel <*insert_3*> .

Severity

0 : Information

Explanation

The channel <*insert_3*> did not supply a certificate to use during SSL handshaking, but a certificate is required by the remote queue manager. The channel did not start.

Response

If the SSL key repository for the queue manager has been specified as '*SYSTEM' ensure that a certificate has been associated with the application description for the queue manager in Digital Certificate Manager. Alternatively, if appropriate, change the remote channel definition so that its SSLCAUTH attribute is set to OPTIONAL and it has no SSLPEER value set.

AMQ9643

Remote SSL peer name error for channel <*insert_3*> .

Severity

30 : Severe error

Explanation

The remote end of channel <*insert_3*> has had an SSL peer name error. The channel did not start.

Response

Review the error logs on the remote system to discover the problem with the peer name.

AMQ9645

Correctly labeled SSL certificate missing on channel <*insert_3*>.

Severity

30 : Severe error

Explanation

The key database file in use has not been set up with a correctly labeled SSL certificate. The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Add a correctly labeled SSL certificate to the current key database file. Restart the channel.

AMQ9646

Channel <*insert_3*> could not connect to any LDAP CRL servers.

Severity

30 : Severe error

Explanation

LDAP Certification Revocation List (CRL) servers were specified but a connection could not be established to any of them. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Check that the LDAP CRL server specifications are correct. If they are, check that the servers are running and that the networking to access them is working correctly. Fix any errors found and restart the channel.

AMQ9647

I/O error on SSL key repository.

Severity

30 : Severe error

Explanation

An I/O error was encountered when attempting to read the SSL key repository. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Analyse why there is a I/O problem when reading the key repository. Fix the error if one is found, or it may be a temporary problem. Restart the channel.

AMQ9648

The SSL key repository has an invalid internal format.

Severity

30 : Severe error

Explanation

The SSL key repository has an invalid internal format. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

re-create the SSL key repository and restart the channel.

AMQ9649

The SSL key repository contains duplicate keys.

Severity

30 : Severe error

Explanation

The SSL key repository contains two or more entries with the same key. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Use your key management tool to remove the duplicate keys. Restart the channel.

AMQ9650

The SSL key repository contains entries with duplicate labels.

Severity

30 : Severe error

Explanation

The SSL key repository contains two or more entries with the same label. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Use your key management tool to remove the duplicate entries. Restart the channel.

AMQ9651
The SSL key repository is corrupt or has a bad password.

Severity
30 : Severe error

Explanation
The SSL key repository has become corrupted or its password id is incorrect. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Use your key management tool to re-create the key repository with a new password. Restart the channel.

AMQ9652
The remote SSL certificate has expired.

Severity
30 : Severe error

Explanation
The SSL certificate used by MQ on the remote end of the channel has expired. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Use your key management tool to provide MQ with a current SSL certificate on the remote end of the channel. Restart the channel.

AMQ9653
An SSL trace file could not be opened.

Severity
10 : Warning

Explanation
An SSL trace file could not be opened. The SSL trace files are created in directory /var/mqm/trace and have names AMQ.SSL.TRC and AMQ.SSL.TRC.1. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. This error is not regarded as sufficiently serious to interrupt channel operation; channel operation was not affected.

Response
Check that you have a directory called /var/mqm/trace and that the userid under which WebSphere MQ runs has permissions and space to create and open a file in that directory. Fix the problem and you will get SSL trace output.

AMQ9654
An invalid SSL certificate was received from the remote system.

Severity
30 : Severe error

Explanation
An SSL certificate received from the remote system was not corrupt but failed validation checks on something other than its ASN fields and date. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Additionally, this error is seen for a certificate validation error 8(ssl_rc) - GSK_ERROR_CERT_VALIDATION. This error occurs when the certificate can not be validated, and the certificate chain can not be built because the certificate is not in the key database.

Response
Ensure that the remote system has a valid SSL certificate. Restart the channel.

AMQ9655
Problem loading GSKit SSL support.

Severity
30 : Severe error

Explanation
MQ SSL support is provided on this platform using a component called GSKit which is installed as part of MQ GSKit had an internal problem loading one if its dynamic link libraries. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Uninstall MQ and reinstall. Restart the channel.

AMQ9656
An invalid SSL certificate was received from the remote system.

Severity
30 : Severe error

Explanation
An SSL certificate received from the remote system was not corrupt but failed validation checks on its ASN fields. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Ensure that the remote system has a valid SSL certificate. Restart the channel.

AMQ9657
The key repository could not be opened (channel <*insert_3*>).

Severity
30 : Severe error

Explanation
The key repository could not be opened. The key repository either does not exist or has incorrect permissions associated with it. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Ensure that the key repository you specify exists and that its permissions are such that the MQ process involved can read from it. Restart the channel.

AMQ9658
An invalid SSL certificate has been encountered.

Severity
30 : Severe error

Explanation
An SSL certificate has been encountered which was not corrupt but which failed validation checks on its date fields. The certificate has either expired, or its date is not valid yet (i.e. the from date is later than today), or the validity date range is incorrect (for example, the to date is earlier than the from date). The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Ensure that both the local and remote systems have valid, current SSL certificates. Restart the channel.

AMQ9659

A failure occurred during SSL handshaking.

Severity

30 : Severe error

Explanation

During SSL handshaking, or associated activities, a failure occurred. The failure is <*insert_4*> and has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Refer to prior message in the WebSphere MQ error log for information related to this problem.

AMQ9660

SSL key repository: password stash file absent or unusable.

Severity

30 : Severe error

Explanation

The SSL key repository cannot be used because MQ cannot obtain a password to access it. Reasons giving rise to this error include:

(a) the key database file and password stash file are not present in the location configured for the key repository,

(b) the key database file exists in the correct place but that no password stash file has been created for it,

(c) the files are present in the correct place but the userid under which MQ is running does not have permission to read them,

(d) one or both of the files are corrupt.

The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Ensure that the key repository variable is set to where the key database file is. Ensure that a password stash file has been associated with the key database file in the same directory, and that the userid under which MQ is running has read access to both files. If both are already present and readable in the correct place, delete and re-create them. Restart the channel.

AMQ9661

Bad SSL data from peer on channel <*insert_3*> .

Severity

30 : Severe error

Explanation

An SSL channel has stopped because bad SSL data was received from the remote end of the channel. More detail on the nature of the corruption can be found from the GSKit return value of <*insert_1*> (the GSKit return values are documented in the MQ manuals). The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'.

Response

Ensure you are connecting to a version of MQ which supports SSL at the remote end of the channel. Check your network between the two ends of the channel, and consider whether any possible causes of message corruption could be present. Fix any problems which may exist and restart the channel.

AMQ9661 (IBM i)

Bad SSL data from peer on channel <*insert_3*> .

Severity
0 : Information

Explanation
An SSL channel has stopped because bad SSL data was received from the remote end of the channel. More detail on the nature of the corruption can be found from the GSKit return value of <*insert_1*> (the GSKit return values are documented in the MQ manuals and also in the GSKSSL member of the H file in library QSYSINC). The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'.

Response
Ensure the remote queue manager and channel listener are running and that you are connecting to a version of MQ which supports SSL at the remote end of the channel. Check your network between the two ends of the channel, and consider whether any possible causes of message corruption could be present. Fix any problems which may exist and restart the channel.

AMQ9662
SSL has encountered something it does not support.

Severity
30 : Severe error

Explanation
This error can arise for a number of reasons:

(a) The platform does not support a particular type of cryptographic hardware, for example, nCipher nFast and Rainbow Cryptoswift are no longer supported.

(b) The cryptographic hardware cryptography has returned an error.

(c) Unsupported X509 General Name format when checking the remote certificate. The GSKit SSL provider incorporated in MQ only supports formats rfc822, DNSName, directoryname, uniformResourceID, and IPAddress. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Check that your cryptographic hardware is supported on your platform and test it to see that it is working correctly. Check that the remote certificates you are using conform to the X509 General Name formats listed. Fix the problem and restart the channel.

AMQ9663
An invalid SSL certificate was received from the remote system.

Severity
30 : Severe error

Explanation
An SSL certificate received from the remote system failed validation checks on its signature. The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Ensure that the remote system has a valid SSL certificate. Restart the channel.

AMQ9664
Bad userid for CRL LDAP server; SSL channel <*insert_3*> .

Severity
30 : Severe error

Explanation
Certification Revocation List (CRL) checking on an LDAP server or servers has been configured

on the local MQ system. The userid information configured for the LDAP server or servers is incorrect. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Check the userid information for the CRL LDAP server or servers you have configured locally. Correct any problems found and restart the channel.

AMQ9665
SSL connection closed by remote end of channel <*insert_3*> .

Severity
30 : Severe error

Explanation
The SSL connection was closed by the remote end of the channel during the SSL handshake. The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Check the remote end of the channel for SSL-related errors. Fix them and restart the channel.

AMQ9666
Error accessing CRL LDAP servers; SSL channel <*insert_3*> .

Severity
30 : Severe error

Explanation
CRL checking on LDAP servers has been configured on the local MQ system. An error was found when trying to access the CRL LDAP servers when validating a certificate from the remote system. Possible causes are:

(a) cannot connect to any of the LDAP servers, or

(b) the certificate issuer's Distinguished Name (DN) is not defined in the DIT of an LDAP server.

The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Check access to the CRL LDAP server(s) you have configured locally. Put right any problems found and restart the channel.

AMQ9667
Bad user name or password for CRL LDAP server; SSL channel <*insert_3*>.

Severity
30 : Severe error

Explanation
Certification Revocation List (CRL) checking on an LDAP server or servers has been configured on the local MQ system. The user name or password information configured for the LDAP server or servers is incorrect. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Check the user name and password information for the CRL LDAP server or servers you have configured locally. Correct any problems found and restart the channel.

AMQ9668
The specified PKCS #11 shared library could not be loaded.

Severity
30 : Severe error

Explanation

A failed attempt was made to load the PKCS #11 shared library specified to MQ in the PKCS #11 driver path field of the GSK_PKCS11 SSL CryptoHardware parameter. The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Ensure that the PKCS #11 shared library exists and is valid at the location specified. Restart the channel.

AMQ9669

The PKCS #11 token could not be found.

Severity

30 : Severe error

Explanation

The PKCS #11 driver failed to find the token specified to MQ in the PKCS #11 token label field of the GSK_PKCS11 SSL CryptoHardware parameter. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Ensure that the PKCS #11 token exists with the label specified. Restart the channel.

AMQ9670

PKCS #11 card not present.

Severity

30 : Severe error

Explanation

A PKCS #11 card is not present in the slot. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Ensure that the correct PKCS #11 card is present in the slot. Restart the channel.

AMQ9671

The PKCS #11 token password specified is invalid.

Severity

30 : Severe error

Explanation

The password to access the PKCS #11 token is invalid. This is specified to MQ in the PKCS #11 token password field of the GSK_PKCS11 SSL CryptoHardware parameter. The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response

Ensure that the PKCS #11 token password specified on GSK_PKCS11 allows access to the PKCS #11 token specified on GSK_PKCS11. Restart the channel.

AMQ9672

An SSL security call failed.

Severity

30 : Severe error

Explanation

An SSPI call to the Secure Channel (Schannel) SSL provider failed. The failure has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Consult the Windows Schannel reference manual to determine the meaning of status <*insert_5*> for SSPI call <*insert_4*>. Correct the failure and if necessary re-start the channel.

AMQ9673

SSL client handshaking failed.

Severity

30 : Severe error

Explanation

During an SSL client's handshaking, an SSPI call to the Secure Channel (Schannel) SSL provider failed. The failure has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Consult the Windows Schannel reference manual to determine the meaning of status <*insert_4*> for SSPI call <*insert_5*>. Correct the failure and if necessary re-start the channel.

AMQ9674

An unknown error occurred during an SSL security call.

Severity

30 : Severe error

Explanation

An unknown error occurred during an SSPI call to the Secure Channel (Schannel) SSL provider. The error may be due to a Windows SSL problem or to a general Windows problem or to invalid WebSphere MQ data being used in the call. The WebSphere MQ error recording routine has been called. The error has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Consult the Windows Schannel reference manual to determine the meaning of status <*insert_5*> for SSPI call <*insert_4*>. If the problem can be resolved using the manual, correct the failure and if necessary re-start the channel. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9675

The requested certificate could not be found.

Severity

30 : Severe error

Explanation

A request for a certificate identified as <*insert_4*> <*insert_5*> in the store <*insert_3*> has failed, because the certificate could not be found. The Windows error code has been set to <*insert_1*>. The WebSphere MQ error recording routine has been called.

Response

Consult the Windows reference manual to determine the meaning of error <*insert_1*> if this value is non-zero. Check to see whether the specified certificate has been copied to the correct certificate store and has not been deleted. Use the WebSphere MQ Explorer administration application to configure certificate store for use with WebSphere MQ. If the problem cannot be resolved, use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or thehttp://www.ibm.com/software/

support/isa/ , to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9676

The Windows cryptographic services library could not be loaded.

Severity

30 : Severe error

Explanation

WebSphere MQ requires crypt32.dll to be available in order to carry out cryptographic functionality. The attempt to load this library returned the Windows error code <*insert_1*>. The WebSphere MQ error recording routine has been called.

Response

Consult the Windows reference manual to determine the meaning of error code <*insert_1*>. Check that the crypt32.dll file is available and not corrupt. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9677

The Windows security services library could not be loaded.

Severity

30 : Severe error

Explanation

WebSphere MQ requires <*insert_3*> to be available in order to run or configure SSL functionality. The attempt to load this library returned the Windows error code <*insert_1*> . The WebSphere MQ error recording routine has been called.

Response

Consult the Windows reference manual to determine the meaning of error code <*insert_1*>. Check that the <*insert_3*> file is available and not corrupt. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9678

The certificate <*insert_4*>/<*insert_5*> already exists in the store <*insert_3*>.

Severity

10 : Warning

Explanation

The certificate store <*insert_3*> already contains the specified certificate, identified by the issuer name of <*insert_4*>, serial number <*insert_5*> . The existing certificate has not been replaced.

AMQ9679

The certificate store <*insert_3*> could not be opened.

Severity

30 : Severe error

Explanation

The certificate store <*insert_3*> could not be opened, and failed with the Windows error code <*insert_1*> . The WebSphere MQ error recording routine has been called.

Response

Consult the Windows reference manual to determine the meaning of error <*insert_1*> if this value is non-zero. Check that either your MQSSLKEYR environment variable (for client connections), or SSLKEYR queue manager attribute (for WebSphere MQ queue managers) has been defined correctly, and that the file path specified is valid. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9680

A problem was encountered with the specified certificate file.

Severity

30 : Severe error

Explanation

A problem occurred when attempting to read the certificate from the file <*insert_3*>. The file may be corrupt or incorrectly formatted. The Windows error code reported is <*insert_1*>. The WebSphere MQ error recording routine has been called.

Response

Ensure that the certificate file is valid and complete, and in one of the file formats supported by WebSphere MQ If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9681

The requested functionality is not supported on this system.

Severity

30 : Severe error

Explanation

An SSL function was attempted that is not supported on this system. a) importing pfx format certificate files with private key data is only supported on Windows 2000 or greater. b) a the security library installed on your system is not of the correct level and does not contain the pre-requisite functions. On pre Windows 2000 systems, Internet Explorer 4.1 or greater must be installed. The WebSphere MQ error recording routine has been called.

Response

If pre-requisite software is missing, please install the necessary levels of software and retry the operation. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9682

The WebSphere MQ SSL library has not been initialized.

Severity
30 : Severe error

Explanation
The WebSphere MQ SSL library 'amqcssln.dll' has been called without it first being initialized by the calling process.

Response
Ensure that the initialization function has been called prior to issuing any amqcssln function calls.

AMQ9683
The private key data for this certificate is not exportable.

Severity
30 : Severe error

Explanation
An attempt has been made to export the private key data from a certificate, but the properties of the certificate will not allow this. WebSphere MQ needs to be able to export private key data when copying personal certificates between certificate stores. The Windows cryptographic API returned the error code <*insert_1*> .

Response
When requesting the certificate from the certificate authority, the private key data must be marked as exportable to enable WebSphere MQ to be able to copy the certificate and private key data into a WebSphere MQ store. The certificate file may need to be requested again to resolve this problem. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9684
A problem occurred while attempting to access the certificate's properties.

Severity
30 : Severe error

Explanation
The certificate issued by <*insert_3*> with serial number <*insert_4*>, or its private key data, appears to be unusable and may be corrupt. The Windows return code <*insert_1*> was generated when attempting to use this certificate. The WebSphere MQ error recording routine has been called.

Response
Consult the Windows reference manual to determine the meaning of error <*insert_1*>. Check that the certificate is valid and has not been corrupted. If it is possible that the certificate or private key data is corrupt, try to remove the certificate from your system and re-import it. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9685
A problem occurred while accessing the registry.

Severity
30 : Severe error

Explanation

An error occurred while attempting to load or unload the personal registry hive (HKEY_LOCAL_USER) for the user who launched this process. The WebSphere MQ error recording routine has been called.

Response

If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9686

An unexpected error occurred while attempting to manage a certificate store.

Severity

30 : Severe error

Explanation

The Windows cryptographic API returned error code <*insert_1*> when calling the function <*insert_3*> for certificate store <*insert_4*>. The error may be due to a certificate store problem or to a general Windows problem or to a problem with a certificate in the store. The WebSphere MQ error recording routine has been called.

Response

Consult the Windows reference manual to determine the meaning of error <*insert_1*>. Check that the certificate store is valid and not corrupt. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9687

The pfx password provided is invalid.

Severity

30 : Severe error

Explanation

The password supplied for importing or copying the certificate is incorrect, and the operation could not be completed.

Response

Make sure the password is correct and try again. If the password has been forgotten or lost, the certificate will need to be regenerated or exported from the original source.

AMQ9688

The private key data for this certificate is unavailable.

Severity

30 : Severe error

Explanation

The private key data associated with this certificate is reported as being present on the system, but has failed, returning the Windows error code <*insert_1*>. The WebSphere MQ error recording routine has been called.

Response

Consult the Windows reference manual to determine the meaning of error code <*insert_1*>. If the

problem can be resolved using the manual, correct the failure and if necessary re-try the operation. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9689

An unknown error occurred deleting the store <*insert_3*> .

Severity

30 : Severe error

Explanation

The WebSphere MQ certificate store for queue manager <*insert_3*> could not be deleted. The filename for the certificate store is <*insert_4*>. The Windows error code has been set to <*insert_1*>. The WebSphere MQ error recording routine has been called.

Response

Consult the Windows reference manual to determine the meaning of error <*insert_1*>. If the problem can be resolved using the manual, correct the failure and if necessary re-try the operation. Check that the store file exists and that other processes (such as queue managers) that may be accessing the store are not running. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9690

The public key in the issuer's certificate has failed to validate the subject certificate.

Severity

30 : Severe error

Explanation

The public key in the issuer's certificate (CA or signer certificate), is used to verify the signature on the subject certificate assigned to channel <*insert_3*>. This verification has failed, and the subject certificate therefore cannot be used. The WebSphere MQ error recording routine has been called.

Response

Check that the issuer's certificate is valid and available, and that it is up to date. Verify with the certificate's issuer that the subject certificate and issuer certificate should still be valid. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9691

The WebSphere MQ MQI library could not be loaded.

Severity

30 : Severe error

Explanation

The library file <*insert_3*> is expected to be available on your system, but attempts to load it have failed with Windows return code <*insert_1*>. The WebSphere MQ error recording routine has been called.

Response

Ensure that the WebSphere MQ <*insert_3*> library file exists and is available on your system. Consult the Windows reference manual to determine the meaning of error code <*insert_1*>. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ , or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9692

The SSL library has already been initialized.

Severity

20 : Error

Explanation

The SSL library has already been initialized once for this process, any changes to SSL attributes will not take affect, and the original values will remain in force.

Response

If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9693

The password provided for the LDAP server is incorrect.

Severity

30 : Severe error

Explanation

One or more of the LDAP servers used for providing CRL information to WebSphere MQ has rejected a login attempt because the password provided is incorrect. The WebSphere MQ error recording routine has been called. The error has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Ensure that the passwords specified in the AuthInfo objects are correct for each server name provided. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9694

The DN syntax provided for an LDAP search is invalid.

Severity

30 : Severe error

Explanation

The distinguished name provided in one or more AuthInfo object definitions is invalid, and the

request to a CRL LDAP server has been rejected. The WebSphere MQ error recording routine has been called. The error has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Verify that the details supplied in the AuthInfo object definitions for this channel are correct. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9695

The username provided for the LDAP server is incorrect.

Severity

30 : Severe error

Explanation

One or more of the LDAP servers used for providing CRL information to WebSphere MQ has rejected a login attempt because the username provided does not exist. The WebSphere MQ error recording routine has been called. The error has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Ensure that the usernamed specified in the AuthInfo objects for this channel are correct for each LDAP server name provided. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9697

WebSphere MQ Services could not be contacted on the target server.

Severity

30 : Severe error

Explanation

An attempt was made to contact the WebSphere MQ Services on the target server <*insert_3*>. The call failed with return code <*insert_1*>. The WebSphere MQ error recording routine has been called.

Response

Ensure that the target server name specified is correct and that you have sufficient access rights on that server to be able to administer WebSphere MQ If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9698

An SSL security call failed during SSL handshaking.

Severity

30 : Severe error

Explanation

An SSPI call to the Secure Channel (Schannel) SSL provider failed during SSL handshaking. The failure has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Consult the Windows Schannel reference manual to determine the meaning of status <*insert_5*> for SSPI call <*insert_4*>. Correct the failure and if necessary re-start the channel.

AMQ9699

An unknown error occurred during an SSL security call during SSL handshaking.

Severity

30 : Severe error

Explanation

An unknown error occurred during an SSPI call to the Secure Channel (Schannel) SSL provider during SSL handshaking. The error may be due to a Windows SSL problem or to a general Windows problem or to invalid WebSphere MQ data being used in the call. The WebSphere MQ error recording routine has been called. The error has caused WebSphere MQ channel name <*insert_3*> to be closed. If the name is '????' then the name is unknown.

Response

Consult the Windows Schannel reference manual to determine the meaning of status <*insert_5*> for SSPI call <*insert_4*>. If the problem can be resolved using the manual, correct the failure and if necessary re-start the channel. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9710

SSL security refresh failed.

Severity

30 : Severe error

Explanation

The request to refresh SSL security was unsuccessful.

Response

Look at previous error messages in the error files to determine the cause of the failure.

AMQ9711

SSL security refresh succeeded but channel restarts failed.

Severity

30 : Severe error

Explanation

The SSL environments for this queue manager have been refreshed so current values and certificates are in use for all SSL channels. However, not all the outbound SSL channels which were running when the security refresh was initiated could be restarted after the refresh had completed.

Response

Look at previous error messages in the error files to determine which channels could not be restarted. Restart these if necessary.

AMQ9712

SSL security refresh timed out waiting for channel <*insert_3*> .

Severity
30 : Severe error

Explanation
The system was performing a security refresh for SSL. This function requests all outbound and inbound SSL channels to stop. It then waits for these channels to actually stop. SSL channel <*insert_3*> did not stop within the timeout period.

Response
Investigate why channel <*insert_3*> is hung. Terminate the hung channel. Rerun the SSL security refresh.

AMQ9713
Channel <*insert_3*> ended: SSL refresh in progress.

Severity
0 : Information

Explanation
The SSL support on this queue manager is in the middle of a security refresh. An attempt was made to start outbound SSL channel <*insert_3*>. It cannot start while the SSL security refresh is in progress. The channel is restarted automatically once the SSL security refresh is complete.

Response
None.

AMQ9714
SSL refresh on receiving queue manager: channel did not start.

Severity
30 : Severe error

Explanation
An SSL security refresh is in progress on the queue manager at the receiving end of this SSL channel. The channel is <*insert_3*> ; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Restart the channel once the SSL refresh is complete. The channel will restart automatically if it is configured to retry the connection.

AMQ9715
Unexpected error detected in validating SSL session ID.

Severity
30 : Severe error

Explanation
This error can arise when the GSKit SSL provider is missing one or more pre-requisite PTFs on the OS/400 platform. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'.

Response
Ensure the GSKit SSL provider is at the latest level of maintenance and restart the channel.

AMQ9716
Remote SSL certificate revocation status check failed for channel <*insert_2*>.

Severity
30 : Severe error

Explanation
WebSphere MQ failed to determine the revocation status of the remote SSL certificate for one of the following reasons:

(a) The channel was unable to contact any of the CRL servers or OCSP responders for the certificate.

(b) None of the OCSP responders contacted knows the revocation status of the certificate.

(c) An OCSP response was received, but the digital signature of the response could not be verified.The details of the certificate in question are<*insert_1*>.

The channel name is <*insert_2*>. In some cases the channel name cannot be determined and so is shown as '????'.
The channel did not start
WebSphere MQ does not allow the channel to start unless the certificate revocation status can be determined.

Response

If the certificate contains an AuthorityInfoAccess extension, ensure that the OCSP server named in the certificate extension is available and is correctly configured.

If the certificate contains a CrlDistributionPoint extension, ensure that the CRL server named in the certificate extension is available and is correctly configured.

If you have specified any CRL or OCSP servers to WebSphere MQ, check that those servers are available and are correctly configured.

Ensure that the local key repository has the necessary SSL certificates to verify the digital signature of the response from the OCSP server.

AMQ9717
Remote SSL certificate revocation status check is unknown for channel <*insert_2*>.

Severity
10 : Warning

Explanation
WebSphere MQ was unable to determine the revocation status of the remote SSL certificate for one of the following reasons:

(a) The channel was unable to contact any of the CRL servers or OCSP responders for the certificate.

(b) None of the OCSP responders contacted knows the revocation status of the certificate.

(c) An OCSP response was received, but the digital signature of the response could not be verified.The details of the certificate in question are<*insert_1*>.

The channel name is <*insert_2*>. In some cases the channel name cannot be determined and so is shown as '????'.
The channel was allowed to start, but the revocation status of the remote SSL certificate has not been checked.

Response

If the certificate contains an AuthorityInfoAccess extension, ensure that the OCSP server named in the certificate extension is available and is correctly configured.

If the certificate contains a CrlDistributionPoint extension, ensure that the CRL server named in the certificate extension is available and is correctly configured.

If you have specified any CRL or OCSP servers to Websphere MQ, check that those servers are available and are correctly configured.

Ensure that the local key repository has the necessary SSL certificates to verify the digital signature of the response from the OCSP server.

If you require certificate revocation checks to be enforced, you should configure WebSphere MQ to require certificate revocation checking. Refer to the security section of the WebSphere MQ product documentation for more information on configuring certificate revocation checking.

AMQ9718

Invalid OCSP URL <*insert_1*>.

Severity

30 : Severe error

Explanation

WebSphere MQ was unable to start an SSL channel because one of the AUTHINFO objects specified in the SSLCRLNL namelist has an invalid OCSPURL parameter.

The OCSP URL is <*insert_1*> and the channel name is <*insert_2*>. In some cases the channel name cannot be determined and so is shown as '????'.

Response

The OCSP URL cannot be blank and must be a valid HTTP URL. Correct the OCSP URL and restart the channel or channel process.

Refer to the security section of the WebSphere MQ product documentation for details of how to use OCSP URLs.

AMQ9719

Invalid CipherSpec for FIPS mode.

Severity

30 : Severe error

Explanation

The user is attempting to start a channel on a queue manager or MQ MQI client which has been configured to run in FIPS mode. The user has specified a CipherSpec which is not FIPS-compliant. The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'.

Response

Redefine the channel to run with a FIPS-compliant CipherSpec. Alternatively, the channel may be defined with the correct CipherSpec and the queue manager or MQ MQI client should not be running in FIPS mode; if this is the case, ensure that FIPS mode is not configured. Once the error is corrected, restart the channel.

AMQ9720

QUEUE MANAGERS:

Severity

0 : Information

Explanation

None.

Response

None.

AMQ9721

Queue Manager Name: <*insert_3*>

Severity

0 : Information

Explanation

None.

Response
None.

AMQ9722

CLIENTS:

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9723

Client Certificate Store: <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9724

Expiry Time: <*insert_1*>

Migration Status: To be migrated

Password: ********

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9725

Expiry Time: <*insert_1*>

Migration Status: Failed

Password: ********

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9726
A certificate failed to be migrated because it has an invalid date.

The certificate's details are:

[Microsoft Certificate Store], [Subject], [Issuer], [Serial Number]:

<*insert_3*> .

Severity
30 : Severe error

Explanation
During the migration of a certificate, the certificate's date fields have been found to be invalid. The certificate has either expired or its "from" date is later than today's date or its "to" date is earlier than the "from" date.

The certificate has not been migrated.

Response
If the certificate is required for migration then obtain a valid replacement before importing it into the GSKit key database <*insert_5*>.

AMQ9727
A certificate failed to be migrated because it has an incomplete certification path.

The certificate's details are:

[Microsoft Certificate Store], [Subject], [Issuer], [Serial Number]:

<*insert_3*>.

Severity
30 : Severe error

Explanation
During the migration of a certificate, the certificate's certificate authority (signer) certificate could not be found. The certificate is therefore regarded as an orphan certificate.

A copy of the certificate has been written to the file name <*insert_4*> .

If file name is suffixed ".cer" then the certificate is a certificate authority (signer) certificate. If file name is suffixed ".pfx" then the certificate is a personal certificate and it has a password which is the same as that specified for the GSKit key database <*insert_5*>. The certificate has not been migrated.

Response
If the certificate is required for migration then ensure that a complete certification path exists in the GSKit key database <*insert_5*> before importing the certificate.

AMQ9728
A certificate failed to be migrated because it could not be imported into the GSKit key database <*insert_5*>.

The certificate's details are:

[Microsoft Certificate Store], [Subject], [Issuer], [Serial Number]:

<*insert_3*> .

Severity
30 : Severe error

Explanation
A certificate failed to be imported because there was a problem during the migration of the certificate.

A copy of the certificate has been written to the file name <*insert_4*> .

If file name is suffixed ".cer" then the certificate is a certificate authority (signer) certificate. If file name is suffixed ".pfx" then the certificate is a personal certificate and it has a password which is the same as that specified for the GSKit key database <*insert_5*>. The certificate has not been migrated.

Response

Refer to the previous message in the error log to determine the cause of the failure. If appropriate, refer to the Windows or GSKit reference documentation to determine the cause.

AMQ9729

Unable to create certificate file <*insert_3*> .

Severity

30 : Severe error

Explanation

A certificate failed to be imported because there was a problem during the migration of the certificate. In addition to this first problem, a second problem occurred when trying to create a copy of the certificate by writing it to the file <*insert_3*> . The certificate is located in the Microsoft Certificate Store <*insert_4*>. The certificate is intended for the GSKit key database <*insert_5*>. If file name is suffixed ".cer" then the certificate is a certificate authority (signer) certificate. If file name is suffixed ".pfx" then the certificate is a personal certificate. The certificate has not been migrated.

Response

Determine the cause of the 2 problems. Refer to the previous message in the error log to determine the cause of the first failure. If appropriate, refer to the Windows or GSKit reference documentation to determine the cause. The second failure occurred during a call to the Windows 'CreateFile' function with a return code of <*insert_1*>. For this failure, check that file does not already exist and that you have authority to create this file.

AMQ9730

Certificate migration has completed with no failures. The number of certificates migrated was <*insert_1*>.

Severity

0 : Information

Explanation

The migration of certificates from the Microsoft Certificate Store <*insert_3*> to the GSKit key database <*insert_4*> has completed and there were no migration failures. The number of certificates migrated was <*insert_1*> .

Response

If any certificates were migrated, use the GSKit iKeyman GUI to verify that the GSKit key database contains all the certificates required to support the intended SSL channel. If no certificates were migrated then this is probably because <*insert_3*> contained only a default set of certificate authority (signer) certificates. The default set is not migrated because the newly created GSKit key database will have its own set which will be the same or more up to date.

Although there were no failures which caused certificates not to be migrated, there may have been other failures and these must be resolved otherwise the SSL channel may subsequently fail to start. Refer to the error log and check for any failures.

AMQ9732

A registry entry already exists for <*insert_3*> .

Severity

30 : Severe error

Explanation

The command has been used to request automatic migration for a queue manager's or a client's Microsoft Certificate Store. However, there is already an entry in the registry for this store. If the request was for a queue manager then <*insert_3*> is the queue manager name, otherwise it is the name of the client's Microsoft Certificate Store.

Response

List, and then check, the contents of the registry by running the Transfer Certificates (amqtcert) command with the options "-a -l". If it is necessary to replace the entry then firstly remove it, by using amqtcert with the "-r" option, then use amqtcert to request automatic migration.

AMQ9733

The request to automatically migrate certificates has completed successfully.

Severity

0 : Information

Explanation

A request was made to automatically migrate SSL certificates. This request may have been made during the installation of WebSphere MQ or by using the Transfer Certificates (amqtcert) command. The request has now been performed and the migration has completed successfully.

Response

Use the GSKit iKeyman GUI to verify that the GSKit key database contains all the certificates required to support the intended SSL channel. If no certificates were migrated then this is because the Microsoft Certificate Store contained only a default set of certificate authority (signer) certificates. The default set is not migrated because the newly created GSKit key database will have its own set which will be the same or more up to date.

AMQ9734

There was a failure during the automatic migration of certificates.

Severity

30 : Severe error

Explanation

A request was made to automatically migrate SSL certificates. This request may have been made during the installation of WebSphere MQ or by using the Transfer Certificates (amqtcert) command. The request has now been performed but there was a failure during the migration process.

Response

Refer to previous messages in the error log to determine the cause of the failure. It may be the case that all certificates have successfully migrated and that the failure did not affect this part of the migration process. In this case, use the GSKit iKeyman GUI to verify that the GSKit key database contains all the certificates required to support the intended SSL channel.

AMQ9735

Certificate migration has terminated unexpectedly. A failure occurred during GSKit initialization.

Severity

30 : Severe error

Explanation

The certificate migration process has terminated unexpectedly. The migration requires the GSKit environment to be successfully initialized. This involves the GSKit operations of initialization, creation of the key database and stashing of the key database password. There was a failure during one of these operations. No certificates have been migrated. If the stashing of the password failed then the key database <*insert_4*> will have been created. The failure occurred during the GSKit operation <*insert_3*> and the GSKit return code <*insert_1*> was generated.

Response

If the key database has been created then, after the cause of the failure has been resolved, delete it, remove the relevant registry state information and then re-try the certificate migration process. Use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at

http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9736

The library <*insert_3*> was not found.

Severity

30 : Severe error

Explanation

An attempt to dynamically load the library <*insert_3*> failed because the library was not found. If this an WebSphere MQ library, it is only available on WebSphere MQ server installations and is required when the Transfer Certificates (amqtcert) command is used to perform a queue manager operation. If this a GSKit library, it should have been installed during the WebSphere MQ installation.

Response

Do not use the command to perform a queue manager operation on a WebSphere MQ MQI client-only installation. If the command has been made on a WebSphere MQ server installation, or if it is a GSKit library which is missing, then record the problem identifier, save any generated output files and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9737

Unable to allocate memory.

Severity

30 : Severe error

Explanation

An attempt to allocate memory failed.

Response

Make more memory available to the command.

AMQ9739

The certificate store <*insert_3*> could not be accessed.

Severity

30 : Severe error

Explanation

The certificate store <*insert_3*> could not be accessed, and failed with Windows error code <*insert_1*> . If you are using the -c parameter check that the name given to amqtcert is correct. If you are using the -m parameter check the SSLKEYR value on the queue manager specified.

Response

Consult the Windows reference manual to determine the meaning of error <*insert_1*> if this value is non-zero. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9740

The certificate store <*insert_3*> could not be opened.

Severity
30 : Severe error

Explanation
The certificate store <*insert_3*> could not be opened, and failed with Windows error code <*insert_1*> .

Response
Consult the Windows reference manual to determine the meaning of error <*insert_1*> if this value is non-zero. If the problem cannot be resolved then use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9741
A problem occurred during a Windows operation.

Severity
30 : Severe error

Explanation
During operation <*insert_3*>, the Windows return code <*insert_1*> was generated.

Response
Consult the Windows reference manual to determine the meaning of return code <*insert_1*> for operation <*insert_3*>.

AMQ9742
A problem occurred during a GSKit operation.

Severity
30 : Severe error

Explanation
During operation <*insert_3*>, the GSKit return code <*insert_1*> was generated.

Response
Use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9743
A certificate failed to be migrated and failed to be logged.

The certificate's details are:

[Microsoft Certificate Store], [Subject], [Issuer], [Serial Number]:

<*insert_3*> .

Severity
30 : Severe error

Explanation
There was a problem trying to migrate a certificate to the GSKit key database <*insert_5*>.

Response
Refer to the previous message in the error log to determine why the migration failed.

AMQ9744
No matching automatic migration registry entry.

Severity
10 : Warning

Explanation
There is no automatic certificate migration entry in the registry which matches the input provided.

Response
None, if the entry was correctly specified. Otherwise, input the command again with correct parameters.

AMQ9745
amqtcert: insufficient memory to migrate certificates.

Severity
30 : Severe error

Explanation
An attempt to allocate memory failed while amqtcert was migrating certificate file <*insert_3*>.sto'. The migration did not complete successfully.

Response
Do not delete <*insert_3*>.sto', but delete all other files called <*insert_4*>.*' (these were created as a result of the failed migration). Also, if you want to rerun this migration automatically, use the -r flag on amqtcert to remove the automatic migration registry entry for this .sto file. Then use the -a flag on amqtcert to create a new automatic migration registry entry for this .sto file.

Make more memory available. Rerun the migration.

AMQ9746
File <*insert_3*> not found.

Severity
30 : Severe error

Explanation
The file specified as a command argument has not been found. The characters ".sto" have been automatically appended to the file name.

Response
Check that file exists and that it is specified as the absolute (rather than relative) directory path and file name (excluding the .sto suffix) of the Microsoft Certificate Store.

AMQ9747
Usage: amqtcert [-a] [-c [Filename | *]] [-e ExpirationTime] [-g FileName]

[-i ListNumber] [-l] [-m [QMgrName | *]] [-p Password]

[-r] [-u ClientLogonID] [-w FileName]

Severity
0 : Information

Response
None.

AMQ9748
A problem occurred accessing the Windows registry.

Severity
30 : Severe error

Explanation

An attempt to access a key or value or data field in the Windows registry key failed. The failure may be due to part of the registry being in an invalid state or may be due to insufficient authority to access that part. The WebSphere MQ error recording routine has been called.

Response

If <*insert_3*> includes the name of a Windows call, consult the Windows reference manual to determine the meaning of status <*insert_1*> for that call. Use the standard facilities supplied with your system to record the problem identifier, and to save the generated output files. Use either the IBM WebSphere MQ support web page at http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9749

Invalid combination of command arguments.

Severity

30 : Severe error

Explanation

The command syntax is incorrect because of an invalid combination of arguments.

Response

Re-try the command using a valid combination of arguments.

AMQ9750

File <*insert_3*> already exists.

Severity

30 : Severe error

Explanation

The file <*insert_3*> cannot be created because it already exists.

Response

Ensure that the file does not exist in the directory. If necessary, make a copy of the file before renaming or moving or deleting it.

AMQ9751

You are not authorized to perform the requested operation.

Severity

30 : Severe error

Explanation

You tried to issue a command for which you are not authorized.

Response

Contact your system administrator to perform the command for you or to request authority to perform the command.

AMQ9752

A certificate failed to be migrated because a Windows operation failed.

The certificate's details are:

[Microsoft Certificate Store], [Subject], [Issuer], [Serial Number]:

<*insert_4*> .

Severity

30 : Severe error

Explanation

A personal certificate could not be migrated because there was a failure during the Windows operation <*insert_3*> with a return code of <*insert_1*>. A personal certificate is exported, with its private key data, from the Microsoft Certificate Store prior to being imported into the GSKit key database. The failure occurred during the export and is probably due to a problem with accessing or using the private key data assoicated with the personal certificate.

Response

Check that the private key data is available and that you have authority to access it. Consult the Windows reference manual to determine the meaning of return code <*insert_1*> for operation <*insert_3*>.

AMQ9753

File <*insert_3*> is empty.

Severity

30 : Severe error

Explanation

The file <*insert_3*> cannot be used because it is empty.

Response

Ensure that the correct file has been used and if necessary investigate the reason for it being empty.

AMQ9754

A certificate failed to be migrated because a GSKit operation failed.

The certificate's details are:

[Microsoft Certificate Store], [Subject], [Issuer], [Serial Number]:

<*insert_4*> .

Severity

30 : Severe error

Explanation

During operation <*insert_3*>, the GSKit return code <*insert_1*> was generated.

Response

Use the standard facilities supplied with your system to record the problem identifier and save the generated output files, and then use either the http://www.ibm.com/support/entry/portal/Overview/Software/WebSphere/WebSphere_MQ, or the IBM support assistant at http://www.ibm.com/software/support/isa/, to see whether a solution is already available. If you are unable to find a match, contact your IBM support center. Do not discard these files until the problem has been resolved.

AMQ9755

Certificate migration has completed with some failures. The number of certificates migrated was <*insert_1*>.

Severity

0 : Information

Explanation

The migration of certificates from the Microsoft Certificate Store <*insert_3*> to the GSKit key database <*insert_4*> has completed but there has been one or more failures. The number of certificates migrated was <*insert_1*> .

Response

If any certificates were migrated, use the GSKit iKeyman GUI to verify that the GSKit key database contains all the certificates required to support the intended SSL channel. The failures

must be resolved otherwise the SSL channel may subsequently fail to start. Refer to previous messages in the error log to determine the cause of such failures.

AMQ9756
The number of certificates in the Microsoft Certificate Store <*insert_3*> is <*insert_1*>.

Severity
0 : Information

Explanation
Provides a count of the number of certificates in the Microsoft Certificate Store <*insert_3*>.

Response
None.

AMQ9757

Certificate <*insert_1*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9758
Subject: <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9759
Issuer: <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9760
Valid From: <*insert_3*> to <*insert_4*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9761
Certificate Usage: <All>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9762
Certificate Usage: <*insert_3*>

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9763
Certificate Type: Personal

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9764
Certificate Type: Signer

Severity
0 : Information

Explanation
None.

Response
None.

AMQ9765
Personal certificate not found for the command option "-i <*insert_1*>".

Severity
30 : Severe error

Explanation
The Transfer Certificates (amqtcert) command was executed using the "-i ListNumber" option with a value of <*insert_1*> . However, no personal certificate was found which corresponded to this value. Certificate migration has failed and no certificates were migrated.

Response
Check that the option value corresponds to a correctly identified personal certificate. If it is not correct then run the command using the "-l List" option to determine the correct number. A GSKit key database, and its associated key database files, was created when the command was run using the "-i ListNumber" option. The database and associated files must be deleted before re-trying the command with the "-i" option.

AMQ9766
A failure occurred creating the GSKit key database <*insert_4*> .

Severity
30 : Severe error

Explanation
GSKit was unable to create the key database and its associated files. During the GSKit operation <*insert_3*> , the return code <*insert_1*> was generated. This is probably due to insufficient authority or to insufficient disk space being available.

Response
Check that you have sufficient authority and that there is sufficient disk space available.

AMQ9767
Usage: strmqikm [iKeymanWorkingDirectory]

Severity
0 : Information

Response
None.

AMQ9768
Directory <*insert_3*> not found.

Severity
30 : Severe error

Explanation
The directory specified as a command argument has not been found.

Response
Check that the directory exists and that it is specified as an absolute (rather than relative) directory path.

AMQ9769
Usage: runmqckm

-keydb -changepw Change the password for a key database

-convert Convert the format of a key database

-create Create a key database

-delete Delete a key database

-stashpw Stash the password of a key database into a file

-list Currently supported types of key database.

-cert -add Add a CA Certificate

-create Create a self-signed certificate

-delete Delete a certificate

-details Show the details of a specific certificate

-export Export a personal certificate and associated private key into a PKCS12 file or a key database

-extract Extract a certificate from a key database

-getdefault Show the default personal certificate

-import Import a certificate from a key database or a PKCS12 file

-list List certificates in a key database

-modify Modify a certificate (NOTE: the only field that my be modified is the trust field)

-receive Receive a certificate

-setdefault Set the default personal certificate

-sign Sign a certificate

-certreq -create Create a certificate request

-delete Delete a certificate request from a certificate request database

-details Show the details of a specific certificate request

-extract Extract a certificate from a certificate request database

-list List all certificate requests in a certificate request database

-recreate re-create a certificate request

-version Display iKeycmd version information

-help Display this help text

Severity
0 : Information

Response
None.

AMQ9770
The SSL key repository password has expired.

Severity
30 : Severe error

Explanation
The SSL key repository cannot be used because the password has expired.

The channel is <*insert_3*>; in some cases its name cannot be determined and so is shown as '????'. The channel did not start.

Response
Use your key management tool to reset the password of the SSL key repository, ensuring that a new password stash file is generated.

AMQ9771
SSL handshake failed.

Severity
30 : Severe error

Explanation
The SSL handshake with host <*insert_3*> failed. The SSL handshake was performed using the Java Secure Socket Extension (JSSE).

Response
The SSLSocketFactory used was <*insert_5*> , where 'default' indicates that the JVM's default SSLSocketFactory was used.

The exception thrown by the <*insert_4*> call was <*insert_1*>. Review the exception message for a description of the failure.

Also examine the error logs at the remote end of the channel. These may contain additional information on why the SSL handshake failed.

AMQ9774
Error accessing the channel authentication table

Severity
30 : Severe error

Explanation
The program could not access the channel authentication table.

Response
A value of <*insert_1*> was returned from the subsystem when an attempt was made to access the channel authentication table.

Contact the systems administrator, who should examine the log files to determine why the program was unable to access the authentication table.

AMQ9776
Channel was blocked by user ID

Severity
30 : Severe error

Explanation
The inbound channel <*insert_3*> was blocked from address <*insert_4*> because the active values of the channel were mapped to a userid which should be blocked. The active values of the channel were <*insert_5*> .

Response
Contact the systems administrator, who should examine the channel authentication records to ensure that the correct settings have been configured.

The command DISPLAY CHLAUTH can be used to query the channel authentication records.

AMQ9777
Channel was blocked

Severity
30 : Severe error

Explanation
The inbound channel <*insert_3*> was blocked from address <*insert_4*> because the active values of the channel matched a record configured with USERSRC(NOACCESS).

The active values of the channel were <*insert_5*> .

Response
Contact the systems administrator, who should examine the channel authentication records to ensure that the correct settings have been configured.

The command DISPLAY CHLAUTH can be used to query the channel authentication records.

AMQ9778
IP address is invalid.

Severity
30 : Severe error

Explanation
The IP address <*insert_3*> was found to be invalid.

Response
The processing of the command is terminated. Reissue the command with the IP address parameter specified correctly.

Refer to the commands section of the WebSphere MQ product documentation for more information on the specification of the IP address parameter.

AMQ9779
IP address range error.

Severity
30 : Severe error

Explanation
The IP address <*insert_3*> contains an invalid range. For example the first number is higher or equal to the second number in the range.

Response
The processing of the command is terminated. Reissue the command with the IP address parameter specified correctly.

Refer to the commands section of the WebSphere MQ product documentation for more information on the specification of the IP address parameter.

AMQ9781
IP address overlaps with previous definition.

Severity
30 : Severe error

Explanation
The IP address <*insert_3*> overlaps an existing IP address <*insert_4*>. For example the first number is higher than or equal to the second number in the range.

Response
The processing of the command is terminated. Reissue the command with an IP address parameter that does not overlap a previous definition or remove the existing record and then reissue the command.

Refer to the commands section of the WebSphere MQ product documentation for more information on the specification of the IP address parameter.

AMQ9782
Remote connection blocked.

Severity
30 : Severe error

Explanation
A connection from IP address <*insert_3*> was blocked because it matched the blocking address rule <*insert_4*> .

Response
Verify that the channel authentication blocking rules are correct. If necessary modify the rules to allow the inbound connection, using the SET CHLAUTH command.

Refer to the commands section of the WebSphere MQ product documentation for more information on the specification of the IP address parameter.

AMQ9783
Channel will run using MCAUSER(<*insert_3*>).

Severity
30 : Severe error

Explanation
No matching channel authentication (CHLAUTH) records were found which matched the given fields. Note that the returned MCAUSER value does not take into account any possible action by a channel security exit.

Response
None.

AMQ9784
Match runcheck found a generic value in <*insert_3*> .

Severity
30 : Severe error

Explanation
Match runcheck found a generic value in <*insert_3*> .

When using MATCH(RUNCHECK) all input fields must not contain generic values.

Response
Reissue the command with all fields containing fully specified values.

AMQ9785
Channel is configured to not use the dead-letter queue.

Severity
30 : Severe error

Explanation

A message cannot be transferred across channel <*insert_5*> from address <*insert_4*> and the channel is configured to not use the dead-letter queue. The reason code is <*insert_1*> and the destination queue is <*insert_3*>.

Response
Either correct the problem that caused the channel to try and write a message to the dead-letter queue or enable the channel to use the dead-letter queue.

AMQ9816
Invalid process name <*insert_3*> provided for TMF/Gateway.

Severity
20 : Error

Explanation
IBM WebSphere MQ client for HP Integrity NonStop Server is unable to enlist with the TMF/Gateway for queue manager <*insert_4*> due to an invalid process name provided in the MQTMF_GATEWAY_NAME environment variable.

Response
Ensure the TMF/Gateway is running and the MQTMF_GATEWAY_NAME environment variable is correctly set to the Guardian process name of the TMF/Gateway.

AMQ9817
No PATHMON process name provided to allow enlisting with TMF/Gateway.

Severity
20 : Error

Explanation

IBM WebSphere MQ client for HP Integrity NonStop Server has detected the presence of a TMF transaction and is attempting to enlist with the TMF/Gateway to allow correct participation of the queue manager in the transaction.

IBM WebSphere MQ client for HP Integrity NonStop Server has been unable to find a process name for the PATHMON process hosting the TMF/Gateway server class for queue manager <*insert_3*> in an mqclient.ini file.

Response

Ensure an mqclient.ini file is available for use by the IBM WebSphere MQ client for HP Integrity NonStop Server containing a TMF stanza providing the Guardian process name of a PATHMON that is hosting a TMF/Gateway server class for queue manager <*insert_3*>.

The mqclient.ini file also requires a TMFGateway stanza providing the server class name to be used for queue manager <*insert_3*>.

Refer to the IBM WebSphere MQ product documentation for further information on using an mqclient.ini file with the IBM WebSphere MQ client for HP Integrity NonStop Server.

AMQ9818

No server class provided to allow enlisting with TMF/Gateway for queue manager <*insert_3*>.

Severity

20 : Error

Explanation

IBM WebSphere MQ client for HP Integrity NonStop Server has detected the presence of a TMF transaction and is attempting to enlist with the TMF/Gateway to allow correct participation of the queue manager in the transaction.

IBM WebSphere MQ client for HP Integrity NonStop Server has been unable to find a server class name in an mqclient.ini file for queue manager <*insert_3*> hosted by PATHMON process <*insert_4*>.

Response

Ensure an mqclient.ini file is available for use by the IBM WebSphere MQ client for HP Integrity NonStop Server which contains a TMFGateway stanza providing the server class name to be used for queue manager <*insert_3*>.

Refer to the IBM WebSphere MQ product documentation for further information on using an mqclient.ini file with the IBM WebSphere MQ client for HP Integrity NonStop Server.

AMQ9819

Error encountered while enlisting with TMF/Gateway for queue manager <*insert_5*>.

Severity

20 : Error

Explanation

IBM WebSphere MQ client for HP Integrity NonStop Server has detected the presence of a TMF transaction and is attempting to enlist with the TMF/Gateway server class <*insert_4*> hosted by PATHMON process <*insert_3*> to allow correct participation of the queue manager in the transaction.

IBM WebSphere MQ client for HP Integrity NonStop Server has encountered an error while establishing contact with the TMF/Gateway. Pathsend error (<*insert_1*>), file system error (<*insert_2*>).

Response

These errors are typically the result of configuration problems with the PATHMON process <*insert_3*> or the server class <*insert_4*>. Refer to the HP NSS TS/MP Pathsend and Server Programming Manual for the appropriate corrective action based on the Pathsend error (<*insert_1*>) and file system error (<*insert_2*>).

AMQ9820

Participation in TMF transactions is not support by queue manager <*insert_3*>.

Severity

20 : Error

Explanation

IBM WebSphere MQ client for HP Integrity NonStop Server has detected the presence of a TMF transaction but IBM WebSphere MQ for z/OS queue manager <*insert_3*> does not support participation in TMF transactions.

Response

The version of z/OS queue manager that you are connecting to does not support the TMF Gateway, please upgrade to a supported release.

AMQ9821
Unable to locate PATHMON process <*insert_3*>.

Severity
20 : Error

Explanation
IBM WebSphere MQ client for HP Integrity NonStop Server is unable to locate PATHMON process <*insert_3*>.

Response

The configuration error may be one of the following:

1. The mqclient.ini TMF stanza contains an invalid process name.
2. The PATHMON process <*insert_3*> is not currently running.

AMQ9822
Unable to locate server class <*insert_4*>.

Severity
20 : Error

Explanation
IBM WebSphere MQ client for HP Integrity NonStop Server is unable to locate server class <*insert_4*> hosted by PATHMON process <*insert_3*>.

Response

The configuration error may be one of the following:

1. The mqclient.ini TMFGateway stanza contains an invalid server class name for queue manager <*insert_5*>.
2. The PATHMON process <*insert_3*> has not been configured with server class <*insert_4*>.
3. Server class <*insert_4*> has not been started or is currently frozen.

AMQ9823
Not authorized to use server class <*insert_4*> hosted by PATHMON process <*insert_3*>

Severity
20 : Error

Explanation
IBM WebSphere MQ client for HP Integrity NonStop Server is not authorized to use server class <*insert_4*> hosted by PATHMON process <*insert_3*>.

Response
Check with your systems administrator to ensure you have the correct access permissions. When confirmed you have the correct access permissions, retry the operation.

AMQ9824
TMF/Gateway server class <*insert_4*> has not been configured appropriately.

Severity
20 : Error

Explanation
The TMF/Gateway server class <*insert_4*> hosted by PATHMON process <*insert_3*> has not been configured appropriately.

Response

The configuration error may be one of the following:

1. The server class has not been configured with TMF enabled.

2. The server class has been configured with MAXLINKS set too low for the number of IBM WebSphere MQ client for HP Integrity NonStop Server applications needing to concurrently enlist with the TMF/Gateway.
3. The server class has been configured with TIMEOUT set too low for the time taken by the TMF/Gateway to process a request. Ideally TIMEOUT should not be set, but if it is then it needs to account for the time taken for the TMF/Gateway's associated remote queue manager to respond.

AMQ9871

Cluster maintenance has been running for <*insert_1*> minutes. Phase <*insert_3*> has so far processed <*insert_2*> records

Severity

0 : Information

Explanation

A queue manager will periodically perform a maintenance cycle to refresh and remove state associated with the clusters that it is a member of. This message gives an indication of the progress that is being made.

Response

For large clusters this maintenance process may take a significant period of time, in such situations this message will be periodically repeated until maintenance has completed. When the maintenance cycle has completed message AMQ9872 will be written to this log.

AMQ9872

Cluster maintenance has completed after <*insert_1*> minutes, <*insert_2*> records were processed

Severity

0 : Information

Explanation

A queue manager will periodically perform a maintenance cycle to refresh and remove state associated with the clusters that it is a member of. This message indicates that that cycle has now completed. This message corresponds with one or more instances of message AMQ9871 previously reported.

Response

This message is for informational purposes only, no user response is required.

AMQ9873

An error occurred while restoring the cluster repository cache, reason=<*insert_1*>

Severity

30 : Severe error

Explanation

An error was detected while restoring the cluster cache. The cluster cache held by this queue manager is now incomplete which may result in inconsistencies in cluster resources visible to and owned by this queue manager. See messages in the queue manager and system error logs for details of the error encountered.

Response

Contact your IBM support center to resolve the problem.

AMQ9874

Repository manager failed due of errors. Retry in <*insert_1*> minutes.

Severity

30 : Severe error

Explanation

Repository manager encountered a problem. See the earlier messages in the queue manager or

system error logs for details. The repository manager will retry the command in <*insert_1*> minutes. If the problem is not rectified no further cluster management activity will occur, this will affect the availability of cluster resources accessed or hosted by this queue manager.

Response

If possible, rectify the identified problem, otherwise contact your IBM support center. Once the problem has been rectified, if the SYSTEM.CLUSTER.COMMAND.QUEUE queue has been set to GET(DISABLED) set the queue to be GET(ENABLED) and wait for the repository manager to retry the command. If the repository manager process has terminated, restart the queue manager.

AMQ9875

REFRESH CLUSTER processing started for cluster.

Severity

0 : Information

Explanation

REFRESH CLUSTER processing started for cluster <*insert_3*> . A REFRESH CLUSTER command has been issued on this queue manager. In phase one this discards all locally cached information for the cluster and requests new information from other members of the cluster when necessary. Phase two processes the information received. For large cluster configurations this process can take a significant time, especially on full repository queue managers, and during this time applications attempting to access cluster resources might see failures to resolve cluster resources. In addition, cluster configuration changes made on this queue manager might not be processed until the refresh process has completed.

Response

Defer any cluster related work on this queue manager until both phases are complete. Message AMQ9442 or message AMQ9404 are issued to this log at the end of phase one. Completion of phase two can be determined when SYSTEM.CLUSTER.COMMAND.QUEUE has reached a consistently empty state.

AMQ9876

Cluster management is about to compress a large number of cache records.

Severity

0 : Information

Explanation

Periodically cluster management will compress its local cache. Compression can take a significant period of time for certain operations, such as performing a CLUSTER REFRESH. During the compression task, cluster management commands will not be processed. Once the compression task has completed message AMQ9877 will be written to this log.

Response

None.

AMQ9877

Cluster cache compression has completed.

Severity

0 : Information

Explanation

A large cache compression has completed. This message corresponds to message AMQ9876 being previously reported.

Response

None.

AMQ9913

The specified local address <*insert_3*> cannot be resolved to an IP address. The return code is <*insert_1*> .

Severity
30 : Severe error

Explanation
An attempt to resolve the local address host name to an IP address has failed.

Response
Check that the local address host name is correct and has an entry in the DNS database.

AMQ9914
The type of local address specified is incompatible with the IP protocol (<*insert_3*>) used.

Severity
30 : Severe error

Explanation
An attempt to use a local address that is incompatible with the IP protocol used.

Response
Make sure that the local address specified is of the same type (IPv4 or IPv6) as the IP Protocol.

AMQ9915
The IP protocol <*insert_3*> is not present on the system.

Severity
30 : Severe error

Explanation
An attempt to use an IP protocol that is not present on the system has been made.

Response
Install the required IP protocol or use an IP protocol that is available on the system. This error can also occur if the system is short of memory or other system resources.

AMQ9920
A SOAP Exception has been thrown.

Severity
30 : Severe error

Explanation
A SOAP method encountered a problem and has thrown an exception. Details of the exception are:

<*insert_3*>

Response
Investigate why the SOAP method threw the exception.

AMQ9921
An error was encountered writing to the Dead Letter Queue.

Severity
30 : Severe error

Explanation
An error was encountered when an attempt was made to write a message to Dead Letter Queue <*insert_3*>. The message was <*insert_4*>.

Response
Ensure that Dead Letter Queue <*insert_3*> exists and is put enabled. Ensure that the Queue Manager attribute DEADQ is set up correctly. Resend the SOAP message.

AMQ9922
Maximum wait time exceeded on queue <*insert_3*> .

Severity
30 : Severe error

Explanation
The maximum time waiting for a message to arrive on queue <*insert_3*> has been exceeded.

Response
Ensure that the queue is not put inhibited. Ensure that messages are being written to the queue.

AMQ9923
Insufficient parameters on command.

Severity
30 : Severe error

Explanation
The SOAP command has been issued with insufficient paramaters.

Response
Supply the correct number of parameters and reissue the command.

AMQ9924
Usage: amqwSOAPNETListener -u WebSphere MQUri

[-w WebServiceDirectory] [-n MaxThreads]

[-d StayAlive] [-i IdContext]

[-x TransactionalControl] [-a Integrity] [-? ThisHelp]

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ9925
Cannot connect to queue manager <*insert_3*> .

Severity
30 : Severe error

Explanation
A SOAP application or the SOAP listener cannot connect to the queue manager <*insert_3*> using <*insert_4*> bindings.

Response
Ensure the bindings are set to the correct value and that the queue manager exists. Check any error messages from the Java MQQueueManager class.

AMQ9926
Null SOAP action specified in a received SOAP message.

Severity
30 : Severe error

Explanation
A NULL soap action has been specified in the SOAP message <*insert_3*>. The message will not be processed.

Response
Include the appropriate SOAP action in the SOAP message.

AMQ9927
MQ queue backout threshold exceeded.

Severity
30 : Severe error

Explanation
The WebSphere MQ backout threshold value has been exceeded for queue <*insert_3*>, processing message <*insert_4*>.

Response
Correct the backout threshold value for queue <*insert_3*> and resend the SOAP message.

AMQ9928
Target service or URI is missing from a SOAP message.

Severity
30 : Severe error

Explanation
The target service or the target URI is missing from SOAP message <*insert_3*>.

Response
Supply a target service or the target URI in the SOAP message.

AMQ9929
Message backout for message (<*insert_3*>) failed.

Severity
30 : Severe error

Explanation
Backout for a message has failed.

Response
Investigate the reason for the backout failure.

AMQ9930
Required Option <*insert_3*> missing from command.

Severity
30 : Severe error

Explanation
The SOAP command was issued with manadatory option <*insert_3*> missing.

Response
Reissue the SOAP command supplying the missing option.

AMQ9931
Invalid value <*insert_3*> specified for option <*insert_4*>.

Severity
30 : Severe error

Explanation
The SOAP command was issued with an invalid value for an option.

Response
Reissue the SOAP command supplying the correct option value.

AMQ9932
Application host class not found

Severity
30 : Severe error

Explanation
Application host class <*insert_3*> has not been found.

Response
Specify the correct application host class in the SOAP message.

AMQ9933
Options <*insert_3*> and <*insert_4*> are mutually exclusive

Severity
30 : Severe error

Explanation
The SOAP command was issued with incompatible options <*insert_3*> and <*insert_4*>.

Response
Reissue the SOAP command supplying compatible options.

AMQ9934
Could not parse URL <*insert_3*>. MQCC_FAILED(2) MQRC_SOAP_URL_ERROR(2212).

Severity
30 : Severe error

Explanation
Could not parse URL <*insert_3*>. MQCC_FAILED(2) MQRC_SOAP_URL_ERROR(2212).

Response
Correct the URL and retry.

AMQ9935
Invalid URL <*insert_3*>. MQCC_FAILED(2) MQRC_SOAP_URL_ERROR(2212).

Severity
30 : Severe error

Explanation
The URL <*insert_3*> failed validation.. MQCC_FAILED(2) MQRC_SOAP_URL_ERROR(2212).

Response
Correct the URL and retry.

AMQ9936
Cannot get connection using <*insert_3*> bindings. MQCC_FAILED(2) MQRC_CONNECTION_ERROR(2273).

Severity
30 : Severe error

Explanation
Cannot get connection using <*insert_3*> bindings. MQCC_FAILED(2) MQRC_CONNECTION_ERROR(2273).

Response
Check that the queue manager is available and running.

AMQ9937
The asyncResult is null. MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR.(2210).

Severity
30 : Severe error

Explanation
The asyncResult is null. MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR.(2210).

Response
Check why the SOAP responses are not being received.

AMQ9938
SOAP/WebSphere MQ Timeout.

Severity
30 : Severe error

Explanation
The MQGET operation timed out. MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR.(2210).

Response
Check why the SOAP responses are not being received. MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR.(2210).

AMQ9939
SOAP/WebSphere MQ Error. MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR.(2210).

Severity
30 : Severe error

Explanation
A SOAP error was detected. MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR.(2210).

Response
Check the WebSphere MQ logs for the reason of the failure.

AMQ9940
Report message returned in MQWebResponse. MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR.(2210).

Severity
30 : Severe error

Explanation
Report message returned in MQWebResponse. MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR.(2210).

Response
Check the report message for the reason of the failure.

AMQ9941
No RFH2 header recognised. MQCC_FAILED(2) MQRCCF_MD_FORMAT_ERROR(3023).

Severity
30 : Severe error

Explanation
No RFH2 header recognised. MQCC_FAILED(2) MQRCCF_MD_FORMAT_ERROR(3023).

Response
Check why the message is being sent with no RFH2 header.

AMQ9942
Message format is not MQFMT_NONE. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Severity
30 : Severe error

Explanation
Message format is not MQFMT_NONE. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Response
Correct the message format and retry.

AMQ9943
Unrecognised RFH2 version. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Severity
30 : Severe error

Explanation
Unrecognised RFH2 version. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Response
Correct the version in the RFH2 message and retry.

AMQ9944
Invalid RFH2 length. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Severity
30 : Severe error

Explanation
Invalid RFH2 length. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Response
Correct the RFH2 length and retry.

AMQ9945
Invalid RFH2 <*insert_3*> folder length. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Severity
30 : Severe error

Explanation
Invalid RFH2 <*insert_3*> folder length. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Response
Correct the RFH2 message and retry.

AMQ9946
Invalid actual message length. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Severity
30 : Severe error

Explanation
Invalid actual message length. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Response
Correct the RFH2 message and retry.

AMQ9947
Invalid RFH2 Folder <*insert_3*> <*insert_4*>. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Severity
30 : Severe error

Explanation
Invalid RFH2 Folder <*insert_3*> <*insert_4*>. MQCC_FAILED(2) MQRC_RFH_FORMAT_ERROR(2421).

Response
Correct the RFH2 folder syntax/format and retry.

AMQ9948
Backout Threshold exceeded. MQCC_FAILED(2) MQRC_BACKOUT_THRESHOLD_REACHED(2362).

Severity
30 : Severe error

Explanation
Backout Threshold exceeded. MQCC_FAILED(2) MQRC_BACKOUT_THRESHOLD_REACHED(2362).

Response
Correct the backout threshold limit and retry.

AMQ9949
<*insert_3*> missing from RFH2. MQCC_FAILED(2) MQRC_RFH_PARM_MISSING(2339).

Severity
30 : Severe error

Explanation
<*insert_3*> missing from RFH2. MQCC_FAILED(2) MQRC_RFH_PARM_MISSING(2339).

Response
Correct the RFH2 message and retry.

AMQ9950
Target service missing from SOAP URL. MQCC_FAILED(2) MQRC_SOAP_URL_ERROR(2212).

Severity
30 : Severe error

Explanation
Target service missing from SOAP URL. MQCC_FAILED(2) MQRC_SOAP_URL_ERROR(2212).

Response
Correct the URL and retry.

AMQ9951
Asynchronous request queued successfully. MQCC_OK(0).

Severity
30 : Severe error

Explanation
Asynchronous request queued successfully. MQCC_OK(0).

Response
Wait for response if any is expected.

AMQ9952
Unexpected message type received. MQCC_FAILED(2) MQRC_UNEXPECTED_MSG_TYPE.(2215).

Severity
30 : Severe error

Explanation
A message of the wrong type was received; for example, a report message was received when one had not been requested.

Response
If you are running WebSphere MQ SOAP using the IBM supplied SOAP/WebSphere MQ sender, please contact IBM. If you are running WebSphere MQ SOAP using a bespoke sender, please check that the SOAP/WebSphere MQ request message has the correct options.

AMQ9953
Either the ContentType or the TransportVersion in the RFH2 header have the wrong value. MQCC_FAILED(2) MQRC_RFH_HEADER_FIELD_ERROR(2228)

Severity
30 : Severe error

Explanation
Either the ContentType or the TransportVersion in the RFH2 header have the wrong value. MQCC_FAILED(2) MQRC_RFH_HEADER_FIELD_ERROR(2228)

Response
Correct the message format and retry.

AMQ9954
ViaTran.Redirect called out of transaction MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR(2410)

Severity
30 : Severe error

Explanation
ViaTran.Redirect called out of transaction MQCC_FAILED(2) MQRC_SOAP_DOTNET_ERROR(2410)

Response
Make sure ViaTran.Redirect is only called in a transaction.

AMQ9955
Usage: amqswsdl [?] Uri inputFile outputFile

Severity
0 : Information

Explanation
This shows the correct usage.

Response
None.

AMQ9990 (IBM i)
Keyword <*insert_3*> not valid for this command or the command is incomplete.

Severity
40 : Stop Error

Explanation
The command is incomplete, or an invalid keyword was specified, or the parameter value of the keyword was not specified.

Response
Complete the command, or correct the keyword, or add the parameter value, and then try the command again.

AMQ9991 (IBM i)
The value specified is not allowed by the command.

Severity
40 : Stop Error

Explanation
<*insert_3*> not valid for parameter <*insert_4*>.

Response
Enter one of the values that is defined for the parameter, and try the command again. More information on parameters and commands can be found in the CL reference manual or the appropriate licensed program manual.

AMQ9992 (IBM i)
A matching parenthesis not found.

Severity
40 : Stop Error

Explanation
A matching left or right parenthesis is missing.

Response
Add the missing parenthesis or remove the extra parenthesis.

AMQ9999
Channel program ended abnormally.

Severity
30 : Severe error

Explanation
Channel program <*insert_3*> ended abnormally.

Response
Look at previous error messages for channel program <*insert_3*> in the error files to determine the cause of the failure. For more information, see Problem determination in DQM.

AMQXR Messages

AMQCO1001E
MQXR service unexpectedly caught communications exception={0}(Exception).

Explanation
An exception was caught by the Communications Manager and it was not able to take a reasonable action in response to the exception.

User action
Investigate and resolve the cause of the underlying exception.

AMQCO1002E
A selection key={0} was found in an unexpected state.

Explanation
A selection key was found in a state that was not expected.

User action
Contact your IBM support center.

AMQCO1003E
Connection={0}(Connection) has insufficient data available to satisfy a get request.

Explanation
The application tried to read more data than is immediately available. After the application has processed the information available to it, it should release control and wait to be called again when more data is available.

User action
Change the application to handle this exception, or use Connection.available() before the get() method is called in order to determine if the get() will succeed.

AMQCO1004E
Connection Close error: {0}.

Explanation
An error occurred when a connection was closed. The session might not have completed normally.

User action
Check that the session completed normally.

AMQCO1005E
SSL key repository file invalid or not found for channel "{1}". The following exception was thrown: {0}

Explanation
The SSL key repository file specified for the channel is not valid.

User action
Check the validity of the specified SSL key repository file.

AMQCO1006I
Channel "{0}" has stopped.

Explanation
The channel has stopped. No further communication with clients will occur on this channel.

User action
No action is required.

AMQCO1007E
Connection "{0}" did not send or receive data for "{1}" milliseconds and has been closed.

Explanation
The application set the idle timer on the connection to {1} milliseconds, but did not send or receive any data within this time, so the connection was closed.

User action
Determine why the connection did not send or receive data and if appropriate set the idleTimer to a longer value.

AMQCO1008E
An SSL Handshake error occurred when a client at "{1}" attempted to connect to channel "{0}": {2}.

Explanation
An error occurred when performing an SSL handshake with a client application. This is often because the client is presenting certificates that the MQXR service does not trust.

User action
Use the information in the exception to diagnose and fix the problem.

AMQCO1009E
An invalid Key Store name="{1}" was specified.

Explanation
The key store name or the pass phrase specified is not valid.

User action
Specify a valid key store file name and password.

AMQCO1010E
An SSL Exception occurred when a client at "{1}" attempted to connect to channel "{0}": {2}.

Explanation
An error occurred when performing an SSL operation with a client application.

User action
Use the information in the exception to diagnose and fix the problem.

AMQCO2001E
An error (probe: {0}) occurred and a Failure Data Capture (FDC) file has been written.

Explanation
A problem was detected and a FDC file was written to aid diagnostics.

User action
Look at the FDC file and attempt to resolve the problem. If the problem cannot be resolved, contact your IBM support center.

AMQCO2002I
Trace is disabled.

Explanation
Tracing the MQXR Service (used in order to diagnose problems) is not currently running.

User action
No action is required.

AMQCO2003I
Trace is enabled.

Explanation
Tracing the MQXR Service (used in order to diagnose problems) is currently running.

User action
No action is required.

AMQCO2004I
"{0}" instances of message "{1}" were suppressed.

Explanation
The number {0} of message identifier "{1}" were suppressed from the log since the last message with this identifier was written.

User action
No additional action is required beyond that for the suppressed message.

AMQCO9999E
{0}

Explanation
If the message does not give sufficient information, check previous messages for further help.

User action
See previous messages for further information.

AMQHT1001E
Invalid text={0}(String) was found in an HTTP request or response.

Explanation
An HTTP request or response contained unexpected data not described in "http://www.w3.org/pub/WWW/Protocols/".

User action
Check that the originator or source of the HTTP request or response is producing valid requests or responses.

AMQHT1002E
HTTP header text={0}(String) was invalid.

Explanation
An HTTP request or response contained unexpected text.

User action
Check that the originator or source of the HTTP request or response is producing valid requests or responses.

AMQHT1003E
Invalid text at location={0} in string={1}(String).

Explanation
A Java Script Object Notation (JSON) string contained unexpected data not described in "http://www.json.org/".

User action
Check that the originator or JSON is producing valid data.

AMQHT2001E
WebSocket Close, status code= {0}

Explanation
The websocket was closed by the remote end.

User action
Examine the Websocket status code and determine why the Websocket was closed if this was not expected.

AMQHT9999E
{0}

Explanation
If the message does not give sufficient information, check previous messages for further help.

User action
See previous messages for further information.

AMQXR0001I
Client {0} disconnected normally.

Explanation
An MQTT disconnect flow was received and processed.

User action
None.

AMQXR0002E
On channel {2}, a throwable {1} resulted when the MQXR service received a message from an MQTT client {0}.

Explanation
Bad data was received from a network connection and could not be processed, the connection is closed by the server.

User action
Determine why the client sent the uninterpretable data.

AMQXR0003I
MQXR JAAS {0} : {1}.

Explanation
The JAAS callback in the MQXR service requested that the message is displayed to the user.

User action
Determine the cause of the security problem described in the text of the message issued by JAAS.

AMQXR0004E
MQSeries verb={0}(String) returned cc={1}(int) {2} rc={3}(int) {4}.

Explanation
A WebSphere MQ verb returned an unexpected reason and completion code.

User action
Look up the reason code to determine what caused the error.

AMQXR0005I
Running {0} version {1}.

Explanation
The command is running.

User action
None.

AMQXR0006E
Invalid argument {0} Usage: runMQXRService -m <queueManagerName> -d <Qmgr Data Directory> -g <MQ Global Data directory>

Explanation
The runMQXRService command arguments are incorrect.

User action
Correct the command.

AMQXR0007E
Invalid argument {0} Usage: endMQXRService -m <queueManagerName> -d <Qmgr Data Directory> -g <MQ Global Data directory>

Explanation
The endMQXRService command arguments are incorrect.

User action
Correct the command.

AMQXR0008E
Exception during start of MQXR service: {0}

Explanation
The MQXR service was starting but encountered a problem. Previous errors or FDCs will provide more detail.

User action
Use previous errors or FDCs to diagnose and address the problem then restart the MQXR service.

AMQXR0009E
Exception during shutdown of MQXR service: {0}

Explanation
The MQXR service was shutting down but encountered a problem. Previous errors or FDCs will provide more detail.

User action
Use previous errors or FDCs to diagnose and address the problem.

AMQXR0010E
An invalid ClientIdentifier {0} was received from "{1}" in an MQTT CONNECT packet on channel {2}.

Explanation
The MQXR service received a ClientIdentifier that is not valid because it contains too few, or too many characters, or the characters are not valid in a queue manager name.

User action
Change the ClientIdentifier to use valid characters.

AMQXR0011E
An error occurred during a publish on topic "{3}" from ClientIdentifier "{0}" UserName "{1}" on channel "{2}". A reason code of "{5}" "{6}" was received during an "{4}" operation.

Explanation
The publication from the client was not able to be completed

User action
Using the reason code, diagnose the cause of the problem, alter the configuration (of the client or the server as appropriate) and then retry the publish.

AMQXR0012E
An error occurred whilst subscribing on topic(s) "{3}" for ClientIdentifier "{0}" userNamer "{1}"on channel "{2}". A reason code of "{5}" "{6}" was received during an "{4}" operation.

Explanation
The subscription from the client was not able to be completed

User action
Using the reason code, diagnose the cause of the problem, alter the configuration (of the client or the server as appropriate) and then reconnect the client and retry the subscription.

AMQXR0013E
Error starting channel "{0}" (on host: "{1}" and port "{2}"). The exception was "{3}".

Explanation
The service was unable to listen for connections on the specified port

User action
Use the exception to diagnose and rectify the problem then restart the affected channel.

AMQXR0014E
Error starting channel "{0}". See earlier errors or FDCs for more details.

Explanation
The service was unable to listen for connections on the specified port because of problems that have been reported in earlier errors or FDCs.

User action
Use the preceding errors or FDCs to diagnose and rectify the problem then restart the affected channel.

AMQXR0015I
MQXR Service started successfully ({0} channels running, {1} channels stopped)

Explanation
The MQXR service has completed the processing that occurs on startup

User action
No action is required.

AMQXR0016I
Channel "{0}" has started

Explanation
This channel is now available for client connections

User action
No action is required

AMQXR0017I
A new channel (called "{0}") has been created

Explanation
In response to a request from a user, a new channel has been created

User action
No action is required

AMQXR0018I
Channel "{0}" has been altered

Explanation
In response to a request from a user, some settings on the channel were changed. Some settings do not take effect until the channel is restarted.

User action
No action is required

AMQXR0019I
Channel "{0}" has been deleted

Explanation
In response to a request from a user, a new channel has been deleted

User action
No action is required

AMQXR0020I
Channel "{0}" has been purged

Explanation
Clients have been disconnected from this channel and state associated with them has been removed

User action
No action is required

AMQXR0021W
Client "{0}" at network address "{1}" disconnected abnormally with exception "{2}".

Explanation
An MQTT client was disconnected from the network for the reason shown by the exception.

User action
Look into the exception cause to determine if action is required.

AMQXR0022I
Client "{0}" previously connected at network address "{1}" now connected at "{2}".

Explanation
A new connection has been made for the client taking over from an existing one.

User action
None, if this was intentional.

AMQXR0023I
Unsupported MQTT protocol version on channel {1}, the exception {0} was thrown.

Explanation
An MQTT client attempted to connect using an unsupported protocol version, the connection is closed by the server.

User action
Reconfigure the client to use a supported protocol version.

AMQXR0024I
A Telemetry Daemon for devices attempted to connect using its private protocol on channel {1}, the exception {0} was thrown.

Explanation
The Telemetry daemon for devices has a private protocol for communication. This protocol is not supported and the connection has been closed by the server.

User action
No user action is required, the daemon should reconnect with a supported protocol. To remove this message, reconfigure the Telemetry daemon for devices to not use the private protocol for this connection.

AMQXR0030W
Invalid Will Message from ClientIdentifier "{0}"

Explanation
The Will Message in the Connect packet is malformed, the client connection is closed by the server.

User action
Check the client application and make sure the will message has a non zero length topic name, and a valid Qos.

AMQXR1001E
MQTTV3Exception message={0}(String).

Explanation
An instance of com.ibm.mqttv3.internal.MQTTException has been caught and wrapped.

User action
Contact your IBM support center.

AMQXR1002E
MQTTV5Exception message={0}(String).

Explanation
An instance of com.ibm.mqtt.encoding.internal.MQTTException has been caught and wrapped.

User action
Contact your IBM support center.

AMQXR1003E
An invalid message type={0}(byte) was received.

Explanation
An invalid MQTT message type was received. The connection is disconnected.

User action
The client connected to the MQXR service is sending invalid MQTT messages. \ Find out what client has connected to the MQXR service and what data it has sent. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, \ contact your IBM support center.

AMQXR1004E
An invalid message version={0}(byte) subVersion={1}(byte) was received.

Explanation
An invalid MQTT message version was received. The connection is disconnected.

User action
The client connected to the MQXR service is sending invalid MQTT messages. Find out what client has connected to the MQXR service and what data it has sent. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, contact your IBM support center.

AMQXR1005E
An invalid message message={0}(Hex) was received.

Explanation
An invalid MQTT message was received. The connection is disconnected.

User action

The client connected to the MQXR service is sending invalid MQTT messages. Find out what client has connected to the MQXR service and what data it has sent. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, contact your IBM support center.

AMQXR10006E

An MQTT message with an invalid MultiByteLength={0}(long) was received.

Explanation

An invalid MQTT message containing an invalid multi-byte length was received. The connection is disconnected.

User action

The MQTT client application might have sent incorrect data, which is interpreted as an incorrect length. Check your MQTT client application, and verify that it is sending correct data. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, contact your IBM support center.

AMQXR1007E

An invalid Attribute type={0}(int) was found.

Explanation

An invalid MQTT attribute was found processing of this message is abandoned and the connection closed.

User action

Gather diagnostics and contact your IBM support center.

AMQXR1008E

An invalid mapped message was detected because of {0}(String).

Explanation

An invalid Mapped message was found, it cannot be processed.

User action

Determine where the message came from and correct the messages so that they are not mapped messages or are created with the correct format.

AMQXR1009E

An invalid WebSocket message was detected because of {0}(String).

Explanation

An invalid WebSocket message was found, it cannot be processed.

User action

Determine where the message came from and correct the messages so that they are correctly formed.

AMQXR1010E

An invalid message qos={0}(int) was received.

Explanation

An invalid MQTT qos was received.

User action

The client connected to the MQXR service is sending invalid MQTT messages. Find out what client has connected to the MQXR service and what data it has sent. Contact the provider of the client code. If you are using a client provided in the WebSphere MQ installation, contact your IBM support center.

AMQXR2001E

The command to end the MQXR service failed connecting to queue manager {0}. Exception: {1}

Explanation
The administrative layer could not connect to the queue manager.

User action
If the queue manager is no longer running, no action is required. If the queue manager is still running, check why the administrative layer is unable to connect.

AMQXR2002E
The command to end the MQXR service failed opening queue {0}. Exception: {1}

Explanation
The administrative layer could not open the queue that is required to send a request end the MQXR service.

User action
Determine why the queue could not be opened and retry stopping the service.

AMQXR2003E
The command to end the MQXR service failed: Failed Operation: {0} Exception ({1}): {2}

Explanation
The administrative layer failed to put or get a message that is required to stop the MQXR service.

User action
Correct the problem and then try stopping the service again.

AMQXR2004E
An error occurred while stopping the MQXR service. Completion Code: {0} Reason: {1}

Explanation
An error occurred while the MQXR service was shutting down.

User action
Use the reason code to diagnose the problem.

AMQXR2005E
An error occurred while releasing queue manager resources. Object: {0} Exception: {1}

Explanation
While cleaning up resources the EndMQXRService command encountered a transient problem.

User action
None.

AMQXR2010E
The MQXR service could not access the file: {0}. Exception: {1}

Explanation
The file is invalid, has an invalid format, or incorrect permissions.

User action
Check the file permissions and ensure the file is valid.

AMQXR2011I
Property {0} value {1}

Explanation
The runMQXRService command has read a property with the assigned value.

User action
None.

AMQXR2012E
Invalid property key={0} value={1}

Explanation
The runMQXRService command read an incorrect properties file.

User action
Look at the property in error, correct it, and reissue the command.

AMQXR2014E
Failed to rename {0} to {1}

Explanation
The file could not be renamed

User action
Look at the permissions on the target file and directory and alter them if necessary

AMQXR2013E
Duplicate authentication methods specified for channel={0}, previous={1} duplicate={2}

Explanation
The runMQXRService command read a properties file that specifies two authentication methods, only one is allowed.

User action
Look at the properties file and locate the definition of the named channel. Correct the file to specify a single authentication method and restart the channel.

AMQXR2014E
The following exception was thrown during the starting of an MQXR channel, channelName = "{0}" : {1}

Explanation
An MQXR channel was starting up but encountered a problem. Previous errors or FDCs will provide more detail.

User action
Use earlier errors or FDCs to diagnose and address the problem then restart the MQXR channel.

AMQXR2015E
The following exception was thrown during the stopping of an MQXR channel, channelName = "{0}" : {1}

Explanation
An MQXR channel was stopping but encountered a problem. Previous errors or FDCs will provide more detail.

User action
Use earlier errors or FDCs to diagnose and address the problem then restart the MQXR channel.

AMQXR2020E
Client {0} attempted to unsubscribe from the topic "{1}" which it is not subscribed to.

Explanation
An MQTT client attempted to unsubscribe from a topic it is not subscribed to.

User action
Check that the application logic is correct, and check for earlier errors that could have caused the application to get into an inconsistent state.

AMQXR2021E
Client {0} attempted to unsubscribe from the queue "{1}" which it is not subscribed to.

Explanation
An MQTT client attempted to unsubscribe from a queue it is not subscribed to.

User action
Check that the application logic is correct, and check for earlier errors that could have caused the application to get into an inconsistent state.

AMQXR2050E
Unable to load JAAS config: {0}. The following exception occurred {1}

Explanation
The JAAS configuration tried to authenticate a user on a connection that was unable to load

User action
Check that the JAAS config selected by the channel exists in the jaas.config file and is valid.

AMQXR2051E
Login failed for ClientIdentifier {0} with exception {1}.

Explanation
The JAAS login failed with the exception shown.

User action
Check that the username and password sent by the client are correct.

AMQXR2053E
Error in a trace factory. The following exception occurred {1}

Explanation
There was a problem starting or stopping trace.

User action
Use the exception to diagnose and rectify the problem and then restart trace.

AMQXR9999E
{0}

Explanation
If the message does not give sufficient information, check previous messages for further help.

User action
See previous messages for further information.

MQJMS Messages

List of messages with message numbers beginning with MQJMS.

Table 265. MQJMS Messages.

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS0000	MQJMS_EXCEPTION_ILLEGAL_STATE	Method "{0}" has been invoked at an illegal or inappropriate time or if the provider is not in an appropriate state for the requested operation.
MQJMS0002	MQJMS_EXCEPTION_INVALID_CLIENTID	WebSphere MQ classes for JMS attempted to set invalid connection's client id.
MQJMS0003	MQJMS_EXCEPTION_INVALID_DESTINATION	Destination not understood or no longer valid.
MQJMS0004	MQJMS_EXCEPTION_INVALID_SELECTOR	WebSphere MQ classes for JMS has given JMS Provider a message selector with invalid syntax.
MQJMS0005	MQJMS_EXCEPTION_MESSAGE_EOF	Unexpected end of stream has been reached when a StreamMessage or BytesMessage is being read.

Table 265. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS0006	MQJMS_EXCEPTION_MESSAGE_FORMAT	WebSphere MQ classes for JMS attempts to use a data type not supported by a message or attempts to read data in the wrong type.
MQJMS0007	MQJMS_EXCEPTION_MESSAGE_NOT_READABLE	WebSphere MQ classes for JMS attempts to read a write-only message.
MQJMS0008	MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE	WebSphere MQ classes for JMS attempts to write a read-only message.
MQJMS0009	MQJMS_EXCEPTION_RESOURCE_ALLOCATION	WebSphere MQ classes for JMS is unable to allocate the resources required for a method.
MQJMS0010	MQJMS_EXCEPTION_TRANSACTION_IN_PROGRESS	Operation invalid because a transaction is in progress.
MQJMS0011	MQJMS_EXCEPTION_TRANSACTION_ROLLED_BACK	Call to Session.commit resulted in a rollback of the current transaction.
MQJMS1000	MQJMS_EXCEPTION_MSG_CREATE_ERROR	Failed to create JMS message.
MQJMS1001	MQJMS_EXCEPTION_UNKNOWN_ACK_MODE	Unknown acknowledge mode "{0}".
MQJMS1004	MQJMS_EXCEPTION_CONNECTION_CLOSED	Connection closed.
MQJMS1005	MQJMS_EXCEPTION_BAD_STATE_TRANSITION	Unhandled state transition from "{0}" to "{1}".
MQJMS1006	MQJMS_EXCEPTION_BAD_VALUE	invalid value for "{0}": "{1}".
MQJMS1007	MQJMS_E_BAD_EXIT_CLASS	failed to create instance of exit class "{0}".
MQJMS1008	MQJMS_E_UNKNOWN_TRANSPORT	unknown value of transportType: "{0}".
MQJMS1009	MQJMS_E_NO_STR_CONSTRUCTOR	no constructor with string argument.
MQJMS1010	MQJMS_E_NOT_IMPLEMENTED	not implemented.
MQJMS1011	MQJMS_E_SECURITY_CREDS_INVALID	security credentials cannot be specified when using MQ bindings.
MQJMS1012	MQJMS_E_NO_MSG_LISTENER	no message listener.
MQJMS1013	MQJMS_E_SESSION_ASYNC	operation invalid whilst session is using asynchronous delivery.
MQJMS1014	MQJMS_E_IDENT_PRO_INVALID_OP	operation invalid for identified producer.
MQJMS1015	MQJMS_E_UNKNOWN_TARGET_CLIENT	unknown value of target client: "{0}".
MQJMS1016	MQJMS_E_INTERNAL_ERROR	an internal error has occurred. Please contact your system administrator. Detail: "{0}".
MQJMS1017	MQJMS_E_NON_LOCAL_RXQ	non-local MQ queue not valid for receiving or browsing.
MQJMS1018	MQJMS_E_NULL_CONNECTION	no valid connection available.
MQJMS1019	MQJMS_E_SESSION_NOT_TRANSACTED	invalid operation for non-transacted session.
MQJMS1020	MQJMS_E_SESSION_IS_TRANSACTED	invalid operation for transacted session.
MQJMS1021	MQJMS_E_RECOVER_BO_FAILED	recover failed: unacknowledged messages might not get redelivered.
MQJMS1022	MQJMS_E_REDIRECT_FAILED	failed to redirect message.
MQJMS1023	MQJMS_E_ROLLBACK_FAILED	rollback failed.
MQJMS1024	MQJMS_E_SESSION_CLOSED	session closed.

Table 265. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS1025	MQJMS_E_BROWSE_MSG_FAILED	failed to browse message.
MQJMS1026	MQJMS_E_EXCP_LSTNR_FAILED	ExceptionListener threw exception: "{0}".
MQJMS1027	MQJMS_E_BAD_DEST_STR	failed to reconstitute destination from "{0}".
MQJMS1028	MQJMS_EXCEPTION_NULL_ELEMENT_NAME	element name is null.
MQJMS1029	MQJMS_EXCEPTION_NULL_PROPERTY_NAME	property name is null.
MQJMS1030	MQJMS_EXCEPTION_BUFFER_TOO_SMALL	buffer supplied by application is too small.
MQJMS1031	MQJMS_EXCEPTION_UNEXPECTED_ERROR	an internal error has occurred. Please contact your system administrator.
MQJMS1032	MQJMS_E_CLOSE_FAILED	close() failed because of "{0}".
MQJMS1033	MQJMS_E_START_FAILED	start() failed because of "{0}".
MQJMS1034	MQJMS_E_MSG_LSTNR_FAILED	MessageListener threw: "{0}".
MQJMS1042	MQJMS_E_DELIVERY_MODE_INVALID	invalid Delivery Mode.
MQJMS1044	MQJMS_E_INVALID_HEX_STRING	String is not a valid hexadecimal number - "{0}".
MQJMS1045	MQJMS_E_S390_DOUBLE_TOO_BIG	Number outside of range for double precision S/390 Float "{0}".
MQJMS1046	MQJMS_E_BAD_CCSID	The character set "{0}" is not supported.
MQJMS1047	MQJMS_E_INVALID_MAP_MESSAGE	The map message has an incorrect format.
MQJMS1048	MQJMS_E_INVALID_STREAM_MESSAGE	The stream message has an incorrect format.
MQJMS1049	MQJMS_E_BYTE_TO_STRING	The WebSphere MQ classes for JMS attempted to convert a byte array to a String.
MQJMS1050	MQJMS_E_BAD_RFH2	The MQRFH2 header has an incorrect format.
MQJMS1051	MQJMS_MSG_CLASS	JMS Message class.
MQJMS1052	MQJMS_E_BAD_MSG_CLASS	Unrecognizable JMS Message class.
MQJMS1053	MQJMS_E_INVALID_SURROGATE	Invalid UTF-16 surrogate detected "{0}".
MQJMS1054	MQJMS_E_INVALID_ESCAPE	Invalid XML escape sequence detected "{0}".
MQJMS1055	MQJMS_E_BAD_TYPE	The property or element in the message has incompatible datatype "{0}".
MQJMS1056	MQJMS_E_UNSUPPORTED_TYPE	Unsupported property or element datatype "{0}".
MQJMS1057	MQJMS_E_NO_SESSION	Message has no session associated with it.
MQJMS1058	MQJMS_E_BAD_PROPERTY_NAME	Invalid message property name: "{0}".
MQJMS1059	MQJMS_E_NO_UTF8	Fatal error - UTF8 not supported.
MQJMS1060	MQJMS_E_SERIALISE_FAILED	Unable to serialize object.
MQJMS1061	MQJMS_E_DESERIALISE_FAILED	Unable to deserialize object.
MQJMS1062	MQJMS_EXCEPTION_HAPPENED	Exception occurred reading message body: "{0}".
MQJMS1063	MQJMS_CHARS_OMITTED	Another "{0}" character(s) omitted.
MQJMS1064	MQJMS_ENCODINGS	Integer encoding: "{0}"=Floating point encoding "{1}".
MQJMS1065	MQJMS_E_COULD_NOT_WRITE	Exception occurred writing message body.
MQJMS1066	MQJMS_E_BAD_ELEMENT_NAME	Invalid message element name: "{0}".

Table 265. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS1067	MQJMS_E_BAD_TIMEOUT	timeout invalid for MQ.
MQJMS1068	MQJMS_E_NO_XARESOURCE	failed to obtain XAResource.
MQJMS1069	MQJMS_E_NOT_ALLOWED_WITH_XA	Not allowed with XASession.
MQJMS1072	MQJMS_E_QMGR_NAME_INQUIRE_FAILED	Could not inquire upon Queue Manager name.
MQJMS1073	MQJMS_E_QUEUE_NOT_LOCAL_OR_ALIAS	Specified MQ Queue is neither a QLOCAL nor a QALIAS.
MQJMS1074	MQJMS_E_NULL_MESSAGE	Unable to process null message.
MQJMS1075	MQJMS_E_DLH_WRITE_FAILED	Error writing dead letter header.
MQJMS1076	MQJMS_E_DLH_READ_FAILED	Error reading dead letter header.
MQJMS1077	MQJMS_E_CONN_DEST_MISMATCH	Connection/destination mismatch.
MQJMS1078	MQJMS_E_INVALID_SESSION	Invalid Session object.
MQJMS1079	MQJMS_E_DLQ_FAILED	Unable to write message to dead letter queue.
MQJMS1080	MQJMS_E_NO_BORQ	No Backout-Requeue queue defined.
MQJMS1081	MQJMS_E_REQUEUE_FAILED	Message requeue failed.
MQJMS1082	MQJMS_E_DISCARD_FAILED	Failure while discarding message.
MQJMS1085	MQJMS_E_RFH_WRITE_FAILED	Error writing RFH.
MQJMS1086	MQJMS_E_RFH_READ_FAILED	Error reading RFH.
MQJMS1087	MQJMS_E_RFH_CONTENTS_ERROR	Unrecognizable or invalid RFH content.
MQJMS1088	MQJMS_E_CC_MIXED_DOMAIN	Mixed-domain consumers acting on the same input is forbidden.
MQJMS1089	MQJMS_E_READING_MSG	Exception occurred reading message body: "{0}".
MQJMS1091	MQJMS_E_UNIDENT_PRO_INVALID_OP	operation invalid for unidentified producer.
MQJMS1093	MQJMS_E_NULL_PARAMETER	A null parameter was passed to the constructor: "{0}".
MQJMS1094	MQJMS_E_INVALID_QUANTITY_HINT	Invalid quantityHint.
MQJMS1096	MQJMS_E_INVALID_MESSAGE_REFERENCE	Invalid MessageReference.
MQJMS1098	MQJMS_E_INVALID_MSG_REF_VERSION	Invalid MessageReference version.
MQJMS1099	MQJMS_E_INVALID_THREAD_VERSION	Invalid MQQueueAgentThread version.
MQJMS1102	MQJMS_E_MULTICAST_NOT_AVAILABLE	Multicast connection cannot be established.
MQJMS1103	MQJMS_E_MULTICAST_LOST_MESSAGES	Lost "{0}" messages in reliable multicast mode.
MQJMS1104	MQJMS_E_MULTICAST_HEARTBEAT_TIMEOUT	Multicast connection disconnected due to timeout.
MQJMS1105	MQJMS_E_MULTICAST_PORT_INVALID	Cannot connect with a specific local port for disthub multicast.
MQJMS1106	MQJMS_DIR_PGM_LIB_NOT_FOUND	Unable to load the native library required for PGM/IP.
MQJMS1110	MQJMS_E_11_NOTSUPPORTED	JMS1.1 Operation not supported by this type.
MQJMS1111	MQJMS_E_11_SERVICES_NOT_SETUP	JMS1.1 The required Queues/Publish Subscribe services are not set up.
MQJMS1112	MQJMS_E_11_INVALID_DOMAIN_SPECIFIC	JMS1.1 Invalid operation for domain specific object.

Table 265. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS1113	MQJMS_E_11_INVALID_CROSS_DOMAIN	JMS1.1 Invalid operation for cross domain object.
MQJMS2000	MQJMS_EXCEPTION_MQ_Q_CLOSE_FAILED	failed to close MQ queue.
MQJMS2001	MQJMS_EXCEPTION_MQ_NULL_Q	MQ Queue reference is null.
MQJMS2002	MQJMS_EXCEPTION_GET_MSG_FAILED	failed to get message from MQ queue.
MQJMS2003	MQJMS_EXCEPTION_QMDISC_FAILED	failed to disconnect queue manager.
MQJMS2004	MQJMS_EXCEPTION_MQ_NULL_QMGR	MQQueueManager reference is null.
MQJMS2005	MQJMS_EXCEPTION_QMGR_FAILED	failed to create MQQueueManager for "{0}".
MQJMS2006	MQJMS_EXCEPTION_SOME_PROBLEM	MQ problem: "{0}".
MQJMS2007	MQJMS_EXCEPTION_PUT_MSG_FAILED	failed to send message to MQ queue.
MQJMS2008	MQJMS_EXCEPTION_MQ_Q_OPEN_FAILED	failed to open MQ queue "{0}".
MQJMS2009	MQJMS_EXCEPTION_MQ_QM_COMMIT_FAILED	MQQueueManager.commit() failed.
MQJMS2010	MQJMS_EXCEPTION_MQ_UNKNOWN_DEFTYPE	unknown value for MQ queue definitionType: "{0}".
MQJMS2011	MQJMS_EXCEPTION_MQ_Q_INQUIRE_FAILED	failed to inquire MQ queue depth.
MQJMS2012	MQJMS_EXCEPTION_XACLOSE_FAILED	XACLOSE failed.
MQJMS2013	MQJMS_EXCEPTION_AUTHENTICATION_FAILED	invalid security authentication supplied for MQQueueManager.
MQJMS2014	MQJMS_EXCEPTION_XACLIENT_FAILED	Queue manager rejected XA client connection.
MQJMS3000	MQJMS_E_TMPQ_FAILED	failed to create a temporary queue from "{0}".
MQJMS3001	MQJMS_E_TMPQ_CLOSED	temporary queue already closed or deleted.
MQJMS3002	MQJMS_E_TMPQ_INUSE	temporary queue in use.
MQJMS3003	MQJMS_E_TMPQ_DEL_STATIC	cannot delete a static queue.
MQJMS3004	MQJMS_E_TMPQ_DEL_FAILED	failed to delete temporary queue.
MQJMS3005	MQJMS_PS_GENERAL_ERROR	Publish/Subscribe failed due to "{0}".
MQJMS3006	MQJMS_PS_TOPIC_NULL	Topic reference is null.
MQJMS3008	MQJMS_PS_COMMAND_MSG_BUILD	Failed to build command "{0}".
MQJMS3009	MQJMS_PS_COMMAND_MSG_FAILED	Failed to publish command to MQ queue.
MQJMS3010	MQJMS_PS_PUBLISH_MSG_BUILD	Failed to build publish message.
MQJMS3011	MQJMS_PS_PUBLISH_MSG_FAILED	Failed to publish message to MQ queue.
MQJMS3013	MQJMS_PS_STORE_ADMIN_ENTRY	Failed to store admin entry.
MQJMS3014	MQJMS_PS_SUB_Q_OPEN_FAILED	Failed to open subscriber queue "{0}".
MQJMS3017	MQJMS_PS_SUB_Q_DELETE_FAILED	Failed to delete subscriber queue "{0}".
MQJMS3018	MQJMS_PS_UNKNOWN_DS	Unknown durable subscription "{0}".
MQJMS3019	MQJMS_E_TMPT_DELETED	TemporaryTopic already deleted.
MQJMS3020	MQJMS_E_TMPT_OUTOFSCOPE	TemporaryTopic out of scope.
MQJMS3021	MQJMS_PS_INVALID_SUBQ_PREFIX	Invalid subscriber queue prefix: "{0}".
MQJMS3022	MQJMS_PS_SUBQ_REQUEUE	Durable re-subscribe must use same subscriber queue; specified:"{0}" original:"{1}".
MQJMS3023	MQJMS_PS_SUB_ACTIVE	Subscription has an active TopicSubscriber.

Table 265. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS3024	MQJMS_PS_NULL_CLIENTID	Illegal use of uninitialized client ID.
MQJMS3025	MQJMS_E_TMPT_IN_USE	TemporaryTopic in use.
MQJMS3026	MQJMS_ERR_QSENDER_CLOSED	QueueSender is closed.
MQJMS3028	MQJMS_PUBLISHER_CLOSED	TopicPublisher is closed.
MQJMS3031	MQJMS_CLIENTID_FIXED	Can't set clientID after connection has been used.
MQJMS3032	MQJMS_CLIENTID_NO_RESET	Resetting the clientID is not allowed.
MQJMS3033	MQJMS_QRECEIVER_CLOSED	QueueReceiver is closed.
MQJMS3034	MQJMS_SUBSCRIBER_CLOSED	TopicSubscriber is closed.
MQJMS3037	MQJMS_MESSAGEPRODUCER_CLOSED	Message Producer is closed.
MQJMS3038	MQJMS_MESSAGECONSUMER_CLOSED	Message Consumer is closed.
MQJMS3039	MQJMS_PS_NULL_NAME	Illegal use of null name.
MQJMS3040	MQJMS_E_BROKER_MESSAGE_CONTENT	Invalid broker control message content: "{0}".
MQJMS3041	MQJMS_E_ALREADY_SET	Field "{0}" already set.
MQJMS3042	MQJMS_E_UNREC_BROKER_MESSAGE	Unrecognizable message from Pub/Sub Broker.
MQJMS3043	MQJMS_E_CLEANUP_REP_BAD_LEVEL	Invalid Level for repeating Cleanup.
MQJMS3044	MQJMS_E_CLEANUP_NONE_REQUESTED	Cleanup level of NONE requested.
MQJMS3045	MQJMS_E_CLEANUP_Q_OPEN_1	Failed to open "{0}": maybe a FORCE or NONDUR level cleanup is running?
MQJMS3046	MQJMS_E_CLEANUP_Q_OPEN_2	Failed to open "{0}": maybe another JMS application is using Pub/Sub with this queue manager?
MQJMS3047	MQJMS_PS_SUBSTORE_NOT_SUPPORTED	Subscription Store type not supported by queue manager.
MQJMS3048	MQJMS_PS_INCORRECT_SUBSTORE	Incorrect Subscription Store type.
MQJMS3049	MQJMS_PS_WRONG_SUBSCRIPTION_TYPE	MQJMS_Messages.MQJMS_PS_ WRONG_SUBSCRIPTION_TYPE = Incorrect Subscription type for this Subscription Store.
MQJMS3050	MQJMS_PS_SUBSCRIPTION_IN_USE	Subscription is already in use and cannot be updated.
MQJMS3051	MQJMS_PS_INVALID_SUB_NAME	Invalid Subscription name.
MQJMS4124	MQJMS_ADMIN_PROPVAL_NULL	Property value for "{0}" is null.
MQJMS4125	MQJMS_ADMIN_INV_PROP	Invalid property for a "{0}": "{1}".
MQJMS4131	MQJMS_ADMIN_OBJTYPE_MISMATCH	Expected and actual object types do not match.
MQJMS5053	MQJMS_UTIL_PS_NO_BROKER	*** No broker response. Please ensure that the broker is running. If you are using the WebSphere MQ broker check that your brokerVersion is set to V1 ***
MQJMS5087	MQJMS_UTIL_PS_INTERNALQ	Unexpected error "{1}" accessing internal queue "{0}".
MQJMS6040	MQJMS_DIR_IMB_BADSOCKNAME	Invalid socket family name: "{0}".
MQJMS6041	MQJMS_DIR_IMB_NOCLASS	An exception occurred while attempting to load socket factory class "{0}", exception: <"{1}">.

Table 265. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS6056	MQJMS_DIR_MIN_NOMORE	Cannot change parameter "{0}" since no more BaseConfig parameter changes are allowed.
MQJMS6057	MQJMS_DIR_MIN_BADSET	Cannot set parameter "{0}" to value "{1}".
MQJMS6058	MQJMS_DIR_MIN_BADGET	error occurred while getting BaseConfig parameter "{0}".
MQJMS6059	MQJMS_DIR_MIN_SECLDERR	An exception occurred while loading the minimal client security implementation.
MQJMS6060	MQJMS_DIR_MIN_UNXEXC	An unexpected exception in minimal client, "{0}".
MQJMS6061	MQJMS_DIR_MIN_BADTOP	A specified topic was malformed, "{0}".
MQJMS6062	MQJMS_DIR_MIN_EOF	EOF was encountered while receiving data in the minimal client.
MQJMS6063	MQJMS_DIR_MIN_BRKERR	The broker indicated an error on the minimal client connection.
MQJMS6064	MQJMS_DIR_MIN_BADMSG	Connector.send was called with an illegal message value.
MQJMS6065	MQJMS_DIR_MIN_BADFIELD	An illegal value was encountered for a field, "{0}".
MQJMS6066	MQJMS_DIR_MIN_INTERR	An unexpected internal error occurred in the minimal client.
MQJMS6067	MQJMS_DIR_MIN_NOTBYTES	A bytes message operation was requested on something that is not a bytes message.
MQJMS6068	MQJMS_DIR_MIN_NOTTEXT	A text message operation was requested on something that is not a text message.
MQJMS6069	MQJMS_DIR_MIN_NOTSTREAM	A stream message operation was requested on something that is not a stream message.
MQJMS6070	MQJMS_DIR_MIN_NOTMAP	A map message operation was requested on something that is not a map message.
MQJMS6071	MQJMS_DIR_MIN_BADBRKMSG	The broker sent an invalid message during authentication.
MQJMS6072	MQJMS_DIR_MIN_UNVPRO	The broker requested an unavailable protocol during authentication.
MQJMS6073	MQJMS_DIR_MIN_AUTHREJ	Minimal client connection rejected because of authentication failure.
MQJMS6074	MQJMS_DIR_MIN_NOQOP	No QOP available in the minimal client.
MQJMS6079	MQJMS_DIR_JMS_NOTHDPOOL	An exception occurred while attempting to load thread pooling support, "{0}".
MQJMS6081	MQJMS_DIR_JMS_FMTINT	An attempt was made to read from a Stream message before a previous read has completed.
MQJMS6083	MQJMS_DIR_JMS_THDEXC	An exception occurred while initializing a thread pool instance, "{0}".
MQJMS6085	MQJMS_DIR_JMS_NEXCLIS	No ExceptionListener has been set.
MQJMS6088	MQJMS_DIR_JMS_KILLMON	The client-side connection monitor is terminating.
MQJMS6090	MQJMS_DIR_JMS_LSTACT	Attempted to synchronously receive on a MessageConsumer for which a listener is active.

Table 265. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS6091	MQJMS_DIR_JMS_TCSTSTP	An IOException occurred when starting or stopping delivery on the connection, "{0}".
MQJMS6093	MQJMS_DIR_JMS_RUNKEXC	An exception occurred during synchronous receive, "{0}".
MQJMS6096	MQJMS_DIR_JMS_INVPRI	A JMSPriority level of "{0}" is outside the range specified in JMS.
MQJMS6097	MQJMS_DIR_JMS_BADID	The specified JMSMessageID, "{0}", is invalid.
MQJMS6105	MQJMS_DIR_JMS_NOMORE	No more client parameter changes allowed.
MQJMS6106	MQJMS_DIR_JMS_BADNUM	An exception occurred when initializing parameter "{0}", exception "{1}".
MQJMS6115	MQJMS_DIR_JMS_TCFLERR	An exception occurred while creating the TopicConnection, "{0}".
MQJMS6116	MQJMS_DIR_JMS_CLOSED	This operation is not permitted on an entity that is closed.
MQJMS6117	MQJMS_DIR_JMS_BDTOPIMPL	The "{0}" implementation of Topic is not supported.
MQJMS6118	MQJMS_DIR_JMS_PBNOWLD	Topic "{0}" contains a wildcard which is invalid for publishing.
MQJMS6119	MQJMS_DIR_JMS_PBIOERR	An IOException occurred while publishing, "{0}".
MQJMS6120	MQJMS_DIR_JMS_TMPVIO	Attempted to use a temporary topic not created on the current connection.
MQJMS6121	MQJMS_DIR_JMS_TSIOERR	An IOException occurred while subscribing, "{0}".
MQJMS6232	MQJMS_DIR_JMS_TSBADMTC	While creating a TopicSubscriber, attempting to add the subscription to the matching engine resulted in the following exception: "{0}".
MQJMS6233	MQJMS_DIR_MTCH_UNKEXC	An unexpected exception was caught in the matching engine: "{0}".
MQJMS6234	MQJMS_DIR_MTCH_NULRM	An attempt was made to remove an object with topic "{0}" from an empty matching engine: "{1}".
MQJMS6235	MQJMS_DIR_MTCH_NULCH	An attempt was made to remove an object with topic "{0}" from the matching engine, but it did not have a cache entry: "{1}".
MQJMS6236	MQJMS_DIR_MTCH_BDTYP	An unknown check type of class "{0}" was encountered in a type-specific matcher.
MQJMS6237	MQJMS_DIR_MTCH_UNKNM	An attempt was made to access an unknown field named "{0}".
MQJMS6238	MQJMS_DIR_MTCH_BDMSG	In attempting to access a field of a message=the following exception occurred: "{0}".
MQJMS6239	MQJMS_DIR_MTCH_ECPREP	An EvalCache get or put operation occurred when the cache was not loaded.
MQJMS6240	MQJMS_DIR_MTCH_ECNMIN	An EvalCache get or put operation specified an invalid id.
MQJMS6241	MQJMS_DIR_MTCH_TOMNY	Too many content attributes were specified.
MQJMS6242	MQJMS_DIR_MTCH_DUPDET	A duplicate MatchTarget was detected in MatchSpace.

Table 265. MQJMS Messages (continued).

List of message numbers, constants and explanatory text for messages beginning with MQJMS.

Message identifier	Message constant	Description
MQJMS6243	MQJMS_DIR_MTCH_NOTPK	An attempt was made to remove MatchTarget "{0}" from MatchSpace, but it has no key (topic).
MQJMS6244	MQJMS_DIR_MTCH_NOSUB	The MatchTarget "{1}" with key (topic) "{0}" could not be removed from MatchSpace because it could not be found.
MQJMS6245	MQJMS_DIR_MTCH_NLTOP	An attempt was made to add a MatchTarget to MatchSpace without a key (topic).
MQJMS6246	MQJMS_DIR_MTCH_BDWLD	An incorrect use of a the topic wildcard character "{0}" was detected.
MQJMS6247	MQJMS_DIR_MTCH_BDSEP	The topic segment separator "{0}" appears in an incorrect position.
MQJMS6248	MQJMS_DIR_MTCH_CNTLD	An error occurred while trying to load or invoke the subscription selector parser.
MQJMS6249	MQJMS_DIR_MTCH_PSTPER	The following exception occurred while parsing a subscription selector: "{0}".
MQJMS6250	MQJMS_DIR_MTCH_BDESC	The escape character was used to terminate the following pattern: "{0}".
MQJMS6251	MQJMS_DIR_MTCH_BDESCL	The escape character "{0}" passed to the pattern tool is longer than one character.
MQJMS6252	MQJMS_DIR_MTCH_UNXTYP	A message field was expected to contain a value of type "{0}" but contained one of type "{1}".
MQJMS6228	MQJMS_DIR_MIN_AUTHEXC	Minimal client authentication failed because exception "{0}".
MQJMS6229	MQJMS_DIR_MIN_QOPDIS	QOP required but disabled for this minimal client.
MQJMS6312	MQJMS_DIR_MIN_NOSUB	Non-authorized subscription to topic "{0}".
MQJMS6311	MQJMS_DIR_MIN_NOXASUP	Transport type 'DIRECT' within a transaction is not supported.
MQJMS6350	MQJMS_DIR_MIN_NOTOBJECT	An object message operation was requested on something that is not an object message.
MQJMS6351	MQJMS_DIR_MIN_TSBADSYN	An exception occurred when creating subscription to <"{0}","{1}">, "{2}".
MQJMS6401	MQJMS_DIR_MIN_PER_NOT_SUPPORTED	Persistent messages not supported for transport type 'DIRECT'.
MQJMS6402	MQJMS_DIR_MIN_TTL_NOT_SUPPORTED	Time to Live > 0 not supported for transport type 'DIRECT'.
MQJMS6403	MQJMS_DIR_MIN_EXP_NOT_SUPPORTED	Topic Expiry > 0 not supported for transport type 'DIRECT'.
MQJMS6404	MQJMS_DIR_MIN_ACK_NOT_SUPPORTED	Client Acknowledge not supported for transport type 'DIRECT'.

Related information:
WMQ JMS Exception Messages

WebSphere MQ Advanced Message Security messages

AMS0000

AMS0001
{0}

AMS1000
The IBM WebSphere MQ Advanced Message Security Java interceptor failed to obtain the security policy. Reason Code: "*{0}*"

Explanation
The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to obtain the security policy.

Action
See subsequent messages in the exception for more details about the cause of the error.

AMS1010
The IBM WebSphere MQ Advanced Message Security Java interceptor failed to unprotect the received message.

Explanation
An error occurred when the IBM WebSphere MQ Advanced Message Security Java interceptor was unprotecting the received message.

Action
See subsequent messages in the exception for more details about the cause of the error

AMS1011
An internal error occurred: the IBM WebSphere MQ Advanced Message Security Java interceptor failed to get the character set and encoding from the incoming message.

Explanation
An error ocurred when IBM WebSphere MQ Advanced Message Security Java interceptor was getting the CCSID and encoding from the incoming message.

Action
Retry the operation. If the problem persists, contact your IBM service representative.

AMS1020
Usage: specify the keystore password and private key password java -cp*{0}* com.ibm.mq.ese.config. KeyStoreConfigProtector keystorepass privkeypass

AMS1030
Failed to retrieve the following system properties: "*{0}*"

Explanation
An error occurred when retrieving certain system properties.

Action
Ensure that the appropriate java permissions are set up in the java.policy for the Java runtime to retrieve these system properties.

AMS1035
Unknown message code: "*{0}*"

Explanation
The text for the message code could not be found in the resource bundles.

Action

Look up information about the displayed message code. Ensure that the appropriate IBM WebSphere MQ language packs are installed on this machine.

AMS1040

Failed to read keystore properties from the keystore configuration file.

Explanation

An error occurred when reading the properties from the keystore configuration file.

Action

Verify that the keystore configuration file is available and that the Java application has read access to this file.

AMS1041

Failed to retrieve the certificate for alias "*{0}*" from the keystore "*{1}*"

Explanation

Certificate for the alias could not be retrieved from the keystore.

Action

Use appropriate certificate management tools to ensure that the keystore contains the certificate for the alias.

AMS1042

Failed to retrieve the certificate for alias "*{0}*" from the keystore "*{1}*"

Explanation

Certificate for the alias could not be retrieved from the keystore.

Action

Use appropriate certificate management tools to ensure that the keystore contains the certificate for the alias.

AMS1043

Failed to retrieve the certificate for alias "*{0}*" from the keystore "*{1}*"

Explanation

Certificate for the alias could not be retrieved from the keystore.

Action

Use appropriate certificate management tools to ensure that the keystore contains the certificate for the alias.

AMS1044

Failed to retrieve the private key for alias "*{0}*" from the keystore "*{1}*"

Explanation

Private key for the alias could not be retrieved from the keystore.

Action

Use appropriate certificate management tools to ensure that the keystore contains the private key for the alias.

AMS1045

Failed to retrieve the private key for alias "*{0}*" from the keystore "*{1}*"

Explanation

Private key for the alias could not be retrieved from the keystore.

Action

Use appropriate certificate management tools to ensure that the keystore contains the private key for the alias.

AMS1046
Failed to retrieve aliases from the keystore: "*{0}*"

Explanation
An error occurred when retrieving aliases from the keystore.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1047
Alias "*{0}*" not found in the keystore "*{1}*"

Explanation
An alias is not found in the keystore.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1048
Failed to retrieve the certificate chain for alias "*{0}*" from the keystore "*{1}*"

Explanation
An error occurred when retrieving the certificate chain for an alias from the keystore.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1049
Failed to verify whether the entry for alias "*{0}*" in the keystore "*{1}*" contains a certificate.

Explanation
An error occurred when verifying whether the entry for an alias in the keystore contains a certificate.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1050
Failed to verify whether the entry for alias "*{0}*" in the keystore "*{1}*" contains a private key.

Explanation
An error occurred when verifying whether the entry for an alias in the keystore contains a private key.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1051
Failed to initialize the keystore "*{0}*"

Explanation
Keystore initialization failed.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1052
Failed to protect the password for alias "*{0}*" in keystore "*{1}*"

Explanation
An error occurred when protecting the password for an alias in the keystore.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1053
Failed to unprotect the password for alias "*{0}*" in keystore "*{1}*"

Explanation
An error occurred when unprotecting the password for an alias in the keystore.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1054
Failed to get the certificates for these recipients: "*{0}*"

Explanation
An error occurred when retrieving certificates for recipients.

Action
Verify that the certificates for these recipients are in the local keystore or in the user registry.

AMS1055
An error occurred when trying to retrieve the recipient certificates.

Explanation
An error occurred when trying to retrieve the recipient certificates.

Action
Look at subsequent messages for details on actions to perform to fix this problem.

AMS1056
The message does not contain an IBM WebSphere MQ Advanced Message Security header or it contains a header that is not valid.

Explanation
The message does not contain an IBM WebSphere MQ Advanced Message Security header or it contains a header that is not valid.

Action
Make sure that the IBM WebSphere MQ Advanced Message Security security policy is the same for the sender and the receiver.

AMS1057
The IBM WebSphere MQ Advanced Message Security header could not be converted from "*{0}*" to UTF8.

Explanation
An error occurred when converting the IBM WebSphere MQ Advanced Message Security header to UTF8.

Action
Make sure this character encoding is supported by your Java runtime. If the problem persists, contact your IBM service representative.

AMS1058
The IBM WebSphere MQ Advanced Message Security header could not be converted from "*{0}*" to UTF8.

Explanation
An error occurred when converting the IBM WebSphere MQ Advanced Message Security header to UTF8.

Action
Make sure this character encoding is supported by your Java runtime. If the problem persists, contact your IBM service representative.

AMS1059
An internal has error occurred. The IBM WebSphere MQ Advanced Message Security header could not be converted to an array of bytes.

Explanation
An internal error occurred when converting the IBM WebSphere MQ Advanced Message Security header to an array of bytes.

Action
Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1060
The format of the recipient name "*{0}*" is not valid.

Explanation
The format of the recipient name is not valid.

Action
Set the extended attribute for the recipients to a valid value.

AMS1061
The specified alias "*{0}*" was not found in the keystore "*{1}*"

Explanation
The alias could not be located on the keystore.

Action
Use appropriate certificate management tools to ensure that the keystore contains the alias specified.

AMS1062
The specified alias "*{0}*" was not found in the keystore "*{1}*"

Explanation
The alias could not be located on the keystore.

Action
Use appropriate certificate management tools to ensure that the keystore contains the alias specified.

AMS1063
The alias "*{0}*" is not a key entry. Keystore: "*{1}*"

Explanation
The alias specified is not a key entry.

Action
Use appropriate certificate management tools to ensure that the alias specified is a key entry.

AMS1064
The keystore password can only contain ASCII characters.

Explanation
The keystore password contains non-ASCII characters.

Action
Change your keystore password to contain only ASCII characters.

AMS1065
Could not read following keys from keystore configuration file: "*{0}*".

Explanation
An error occurred when reading properties from the keystore configuration file.

Action
Verify that the property in the keystore configuration file has correct value.

AMS1066
The PKCS11 keystore initialized succesfully, PKCS11 configuration: "*{0}*".

Explanation
The PKCS11 hardware based keystore has been successfully initialized.

Action
No action is required.

AMS1100
The IBM WebSphere MQ Advanced Message Security interceptor could not parse the keystore configuration file.

Explanation
The IBM WebSphere MQ Advanced Message Security interceptor could not parse the keystore configuration file.

Action
Make sure the keystore configuration file contains all required keys and does not contain duplicate keys.

AMS1101
The IBM WebSphere MQ Advanced Message Security keystore configuration file contains duplicate key: "*{0}*".

Explanation
The IBM WebSphere MQ Advanced Message Security keystore configuration file contains duplicate key.

Action
Make sure the keystore configuration file contains all required keys and does not contain duplicate keys.

AMS1102
The IBM WebSphere MQ Advanced Message Security keystore configuration file must contain absolute path "*{0}*".

Explanation
IBM WebSphere MQ Advanced Message Security expected absolute path but found relative one.

Action
Make sure the keystore configuration file contains absolute path.

AMS1120
An internal error occurred: the quality of protection received by the IBM WebSphere MQ Advanced Message Security Java interceptor is not valid. Quality of protection: "*{0}*"

Explanation
The quality of protection received by the Java interceptor is not valid.

Action
Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1121
An internal error occurred: the quality of protection received by the IBM WebSphere MQ Advanced Message Security Java interceptor is not valid. Quality of protection: "*{0}*"

Explanation
The quality of protection received by the Java interceptor is not valid.

Action
Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1122
An internal error occurred: the encryption strength "*{0}*" received by the IBM WebSphere MQ Advanced Message Security Java interceptor is not valid.

Explanation
The encryption strength received by the Java interceptor is not valid.

Action
Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1123
An internal error occurred: the signature algorithm "*{0}*" received by the IBM WebSphere MQ Advanced Message Security Java interceptor is not valid.

Explanation
The signature algorithm received by the Java interceptor is not valid.

Action
Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1124
An internal error occurred: the signature algorithm "*{0}*" received by the IBM WebSphere MQ Advanced Message Security Java interceptor is not valid.

Explanation
The signature algorithm received by the Java interceptor is not valid.

Action
Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1125
The IBM WebSphere MQ Advanced Message Security Java interceptor failed to protect message.

Explanation
The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to protect the message.

Action
See subsequent messages in the exception for more details about the cause of the error.

AMS1126
The IBM WebSphere MQ Advanced Message Security Java interceptor failed to protect message.

Explanation
The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to protect the message.

Action
See subsequent messages in the exception for more details about the cause of the error.

AMS1127
An internal error occurred: the IBM WebSphere MQ Advanced Message Security Java interceptor found more than one sender certificate in the protected message.

Explanation
Only one sender certificate is expected in the protected message.

Action
Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1128
An internal error occurred: the IBM WebSphere MQ Advanced Message Security Java interceptor failed to find the certificate of the sender in the protected message.

Explanation

The Java interceptor failed to find the certificate of the sender. The protected message is expected to contain the certificate of the sender.

Action

Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1129

An internal error occurred: the syntax of the protected message received by IBM WebSphere MQ Advanced Message Security Java interceptor is not valid.

Explanation

The syntax of the protected message received by the Java interceptor is not valid.

Action

Ensure that your security policy is specified correctly and retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1130

The IBM WebSphere MQ Advanced Message Security Java interceptor failed to unprotect message.

Explanation

The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to unprotect the message.

Action

See subsequent messages in the exception for more details about the cause of the error.

AMS1131

An internal error occurred: the IBM WebSphere MQ Advanced Message Security Java interceptor failed to find any recipients from the received message.

Explanation

The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to obtain any recipients from the received message.

Action

Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1132

The certificate of the sender with the subject name "*{0}*" is not valid.

Explanation

The certificate of the sender is not valid.

Action

See subsequent messages in the exception for more details about the cause of the error.

AMS1133

The certificate of the sender with the subject name "*{0}*" is not valid.

Explanation

The certificate of the sender is not valid.

Action

See subsequent messages in the exception for more details about the cause of the error.

AMS1134

The certificate of the recipient with the subject name "*{0}*" is not valid.

Explanation

The certificate of the recipient is not valid.

Action
See subsequent messages in the exception for more details about the cause of the error.

AMS1135
The protected message type does not match the quality of protection (QOP) specified on the objectspace. The expected QOP is "*{0}*" whereas the actual one is "*{1}*"

Explanation
The protected message type does not match the QOP specified on the object space.

Action
Verify that the security policy is correctly specified.

AMS1136
The IBM WebSphere MQ Advanced Message Security Java interceptor failed to decrypt the protected message.

Explanation
The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to decrypt the protected message.

Action
See subsequent messages in the exception for more details about the cause of the error.

AMS1137
The IBM WebSphere MQ Advanced Message Security Java interceptor failed to verify the protected message signature.

Explanation
The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to verify the protected message signature.

Action
See subsequent messages in the exception for more details about the cause of the error.

AMS1138
The IBM WebSphere MQ Advanced Message Security Java interceptor failed to verify the certificate trust chain. The certificate subject name: "*{0}*"

Explanation
The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to verify the certificate trust chain.

Action
See subsequent messages in the exception for more details about the cause of the error.

AMS1139
The protected message encryption mismatch. The expected encryption strength is "*{0}*" whereas the actual encryption strength is "*{1}*"

Explanation
The protected message encryption does not match the one specified on the object space.

Action
Verify that the security policy is correctly specified.

AMS1140
The receiver of this encrypted message is not on the message recipient list "*{0}*"

Explanation
The certificate of a user that is receiving a message is not on the message RecipientsInfo list.

Action
Verify that the user is on a recipients list in a security policy definition.

AMS1200

The certificate with the following subject name "*{0}*" is not yet valid. The certificate will become valid after "*{1}*"

Explanation

The certificate is not yet within its validity period.

Action

Retry the failing operation as soon as the certificate is valid or modify your configuration to use a valid certificate.

AMS1201

The certificate with the following subject name "*{0}*" is expired. The expiration date of the certificate is "*{1}*"

Explanation

The certificate is expired.

Action

Modify your configuration to use a valid certificate.

AMS1202

The IBM WebSphere MQ Advanced Message Security Java interceptor failed to verify the validity period of the certificate. The subject name of the certificate is "*{0}*"

Explanation

The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to verify the validity period of a certificate.

Action

Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1203

A key usage bit that is not valid was found on the certificate with the following subject name "*{0}*". The "*{1}*" key usage bit should be set to "*{2}*" instead of "*{3}*"

Explanation

A key usage bit that is not valid was found on the certificate.

Action

Modify your configuration to use a certificate that has a valid key usage extension. Consult the Administration guide for more details about certificate settings.

AMS1204

An internal error occurred: the IBM WebSphere MQ Advanced Message Security Java interceptor failed to obtain the encryption algorithm name.

Explanation

An internal error occurred: the IBM WebSphere MQ Advanced Message Security Java interceptor failed to obtain the encryption algorithm name.

Action

Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1205

The IBM WebSphere MQ Advanced Message Security Java interceptor failed to obtain any recipient certificate.

Explanation

The IBM WebSphere MQ Advanced Message Security Java interceptor attempted to encrypt a message, but it did not find certificates of recipients

Action

Make sure that a keystore contains all certificates specifed in the appropriate security policy.

AMS1206
The IBM WebSphere MQ Advanced Message Security Java interceptor failed to obtain any recipient certificate.

Explanation
The IBM WebSphere MQ Advanced Message Security Java interceptor attempted to encrypt a message, but it did not find certificates of recipients

Action
Make sure that a keystore contains all certificates specifed in the appropriate security policy.

AMS1207
An internal error occurred: the IBM WebSphere MQ Advanced Message Security Java interceptor failed to obtain the signature algorithm name.

Explanation
An internal error occurred: the IBM WebSphere MQ Advanced Message Security Java interceptor failed to obtain the signature algorithm name.

Action
Retry the failing operation. If the problem persists, contact your IBM service representative.

AMS1208
No proper key bit was found for the certificate for the subject name "*{0}*". The actual values are: "*{1}*", the proper values are: "*{2}*", the state of at least one bit should match.

Explanation
A key usage bit that is not valid was found on the certificate.

Action
Modify your configuration to use a certificate that has a valid key usage extension. Consult the Administration guide for more details about certificate settings.

AMS1209
IBM WebSphere MQ Advanced Message Security Java interceptor failed to verify CRL signature signed by "*{0}*"

Explanation
IBM WebSphere MQ Advanced Message Security Java interceptor failed to verify CRL signed by the given issuer DN

Action
Make sure certificate of the issuer is present in the local keystore.

AMS1210
The Certificate Revocation List "*{0}*" could not be loaded

Explanation
The CRL cannot be found or accessed.

Action
Modify your configuration to point to a valid CRL. Make sure the CRL can be read.

AMS1211
The IBM WebSphere MQ Advanced Message Security Java interceptor could not access Certificate Revocation List.

Explanation
The CRL cannot be found or accessed.

Action
Modify your configuration to point to a valid CRL. Make sure the CRL can be read.

AMS1212

The IBM WebSphere MQ Advanced Message Security Java interceptor failed to validate the certificate. A certificate with BasicConstraint CA set to true cannot be used as End Entity. The subject name is "*{0}*"

Explanation

The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to verify the certificate.

AMS1213

The IBM WebSphere MQ Advanced Message Security Java interceptor failed to validate the certificate. A certificate with the subject name "*{0}*" has been revoked.

Explanation

The IBM WebSphere MQ Advanced Message Security Java interceptor was not able to verify the certificate.

AMS1300

IBM WebSphere MQ Advanced Message Security internal error: queue manager information could not be saved.("*{0}*")

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor could not save the queue manager connection (hconn) information because an internal error occurred.

Action

If the problem occurs persistently, contact your IBM service representative.

AMS1310

IBM WebSphere MQ Advanced Message Security could not resolve current queue name from the object handle.("*{0}*")

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor could not resolve the current queue name from the object handle. The queue might be opened by some internal WebSphere MQ application program interface (API) other than Message Queue Interface (MQI).

Action

Make sure the application does not use internal WebSphere MQ APIs to open the queue. If the problem persists, contact your IBM service representative.

AMS1311

MQOPEN() call failed: reason code ("*{0}*").

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor call to the WebSphere MQ MQOPEN() function failed with the indicated error.

Action

Consult the WebSphere MQ documentation for an explanation of the error code and suggested corrective action. Ensure that the queue manager is operational, the queue exists.

AMS1312

MQCLOSE() call failed, reason code ("*{0}*").

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor call to the WebSphere MQ MQCLOSE() function failed with the indicated WebSphere MQ reason code.

Action

Consult the IBM WebSphere MQ documentation for an explanation of the error code and

suggested corrective action. Ensure that the queue manager is operational, protected object space are present and correct. Make sure that IBM WebSphere MQ Advanced Message Security are configured and running correctly.

AMS1313

IBM WebSphere MQ Advanced Message Security internal error: message could not be protected because the specified signature algorithm "*{0}*" is not valid.

Explanation

The unexpected signature algorithm has been specified

Action

This is an internal error. Contact your IBM service representative

AMS1325

IBM WebSphere MQ Advanced Message Security internal error: queue information could not be resolved from the current queue object handle. ("*{0}*")

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor could not resolve queue information from the current queue object handle because the current queue was not opened via IBM WebSphere MQ Advanced Message Security, or it was closed.

Action

Make sure that the queue has not already been opened by another WebSphere MQ application and that it has not been previously closed. If the problem persists, contact your IBM service representative.

AMS1326

IBM WebSphere MQ Advanced Message Security internal error: could not find local queue manager CodedCharSetId ("*{0}*").

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor could not find the local queue manager's CodedCharSetId because an internal error occurred.

Action

Make sure that the queue manager is connected through IBM WebSphere MQ Advanced Message Security. If the problem persists, contact your IBM service representative.

AMS1327

Quality of protection "*{0}*" for queue "*{1}*" is invalid

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor detected that the quality of protection specified in the security policy for the queue is invalid.

Action

Make sure that the encryption and signature algorithms specified for the queue in the IBM WebSphere MQ Advanced Message Security security policy definition have valid values.

AMS1328

Message with no protection has been put into the queue "*{0}*".

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor has successfully put a message with the 'none' protection level onto the selected queue.

Action

No action is required.

AMS1329

Message with integrity protection has been put into the queue "*{0}*".

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor has successfully put a message with the 'integrity' protection level onto the selected queue.

Action

No action is required.

AMS1330

Message with privacy protection has been put into the queue "*{0}*".

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor has successfully put a message with the 'privacy' protection level onto the selected queue.

Action

No action is required.

AMS1331

IBM WebSphere MQ Advanced Message Security internal error: could not inquire about queue manager properties ("*{0}*").

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor could not inquire about queue manager's properties because an internal error occurred.

Action

Make sure that the queue manager is connected through IBM WebSphere MQ Advanced Message Security. If the problem persists, contact your IBM service representative.

AMS1340

IBM WebSphere MQ Advanced Message Security internal error: queue information could not be resolved from the current queue object handle. ("*{0}*")

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor could not resolve queue information from the current queue object handle because the current queue was not opened by IBM WebSphere MQ Advanced Message Security, or it was closed.

Action

Make sure that the queue is not already been opened by another WebSphere MQ application, and that it has not been previously closed. If the problem persists, contact your IBM service representative.

AMS1341

IBM WebSphere MQ Advanced Message Security found a valid 'PDMQ' format header in the current message.

Explanation

IBM WebSphere MQ Advanced Message Security interceptor found a valid IBM WebSphere MQ Advanced Message Security header in the current message.

Action

No action is required.

AMS1342

The 'PDMQ' format header from the current message is invalid.

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor did not find a valid IBM WebSphere MQ Advanced Message Security header in the the current message.

Action

Check the Quality of Protection (QoP) setting for the queue object. If the QoP setting for the queue is not 'none', make sure that no IBM WebSphere MQ plain text messages are routed to this queue.

AMS1343

Message quality of protection (''*{0}*'') does not match the quality of protection (''*{1}*'') set for the queue ''*{2}*''.

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor detected a quality of protection (QoP) mismatch between the queue and a message in the queue. QoP mismatches usually occur because the queue QoP is changed between the message put and get.

Action

Make sure that the same queue quality of protection is used for the message put and get. When the queue QoP is changed, clean up all messages in the queue before resuming normal operation.

AMS1344

''*{0}*'' - message was signed by ''*{1}*'' at ''*{2}*'' using ''*{3}*''.

Explanation

This is an informational message used to indicate that a message was signed, and to display the signer's name, message timestamp and signature algorithm used.

Action

No action is required.

AMS1345

''*{0}*'' - message was signed and encrypted by ''*{1}*'' at ''*{2}*'' using ''*{3}*'' and ''*{4}*''.

Explanation

This is an informational message used to indicate that a message was signed and encrypted, and to display the signer's name, message timestamp, signature algorithm name and encryption algorithm name.

Action

No action is required.

AMS1346

Message does not have a valid protection type.

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor detected an invalid protection type in a message header. This usually occurs because the IBM WebSphere MQ message header is not valid.

Action

Retry the operation. If the problem persists, contact your IBM service representative.

AMS1347

The IBM WebSphere MQ Advanced Message Security interceptor has put a defective message on error handling queue ''*{0}*''.

Explanation

This is an informational message that indicates the IBM WebSphere MQ Advanced Message Security put a message it could not interpret on the specified error handling queue.

Action

Make sure only valid messages are put onto queues protected by IBM WebSphere MQ Advanced Message Security.

AMS1348

The IBM WebSphere MQ Advanced Message Security interceptor failed to put a defective message on error handling queue. MQ reason code (''*{0}*'')

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor was unable to put a message it could not interpret on the error handling queue because the indicated IBM WebSphere MQ error occurred.

Action

Consult the IBM WebSphere MQ documentation for more information about the reason code. If the problem persists, contact your IBM service representativ.

AMS1349

IBM WebSphere MQ Advanced Message Security internal error: message could not be converted from source CCSID ''*{0}*'' to target CCSID ''*{1}*''. IBM WebSphere MQ compcode ''*{2}*'' : reason ''*{3}*''.

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor could not convert the message from the source Coded Character Set Identifier (CCSID) to target CCSID.

Action

Consult the IBM WebSphere MQ documentation for the completion code and reason code, take corrective action. If the problem persists, contact your IBM service representative.

AMS1350

IBM WebSphere MQ Advanced Message Security internal error: message could not be retrieved from the queue. MQGET() failed. IBM WebSphere MQ compcode ''*{0}*'' : reason ''*{1}*''.

Explanation

The IBM WebSphere MQ Advanced Message Security interceptor could not get the message from the queue.

Action

Consult the IBM WebSphere MQ documentation for the completion code and reason code, take corrective action. If the problem persists, contact your IBM service representative.

AMS1351

IBM WebSphere MQ Advanced Message Security internal error: unprotected message size ''*{0}*'' bytes does not match the original message size ''*{1}*'' bytes.

Explanation

After unprotecting, the message size does not match the original message size. The message header might have been corrupted or tampered with.

Action

Check the message which has been put on the dead letter queue and the audit logs to find the cause of the error and the origin of the message. If the problem persists, contact your IBM service representative.

AMS1352

IBM WebSphere MQ Advanced Message Security internal error: unprotected message QoP does not match QoP indicated in the header. Queue manager is ''*{0}*'', queue is ''*{1}*'', msgId is ''*{2}*''

Explanation

During unprotecting the mismatch between the message buffer quality of protection type and the one specified by the IBM WebSphere MQ Advanced Message Security header was discovered. The message header might have been corrupted or tampered with.

Action

Check the message which has been put into the dead letter queue and the audit logs to find the cause of the error and the origin of the message. If the problem persists, contact your IBM service representative.

AMS1353
Message with quality of protection (''*{0}*'') higher than (''*{1}*'') set for the queue ''*{2}*'' has been accepted.

Explanation
The IBM WebSphere MQ Advanced Message Security interceptor detected a quality of protection (QoP) mismatch between the queue and a message in the queue. QoP mismatch usually occurs when the QoP of the queue is changed between the message put and get.

Action
Make sure that the same queue quality of protection is used for the message put and get. When the QoP of the queue is changed, remove all messages from the queue before resuming any operation.

AMS1354
Message signer is not in the list of authorised signers.

Explanation
The IBM WebSphere MQ Advanced Message Security interceptor detected that the message is signed by an unauthorised party.

Action
Make sure the sender is mentioned in the list of allowed signers.

AMS1355
Message could not be moved from queue (''*{0}*'') to error handling queue. IBM WebSphere MQ completion code (''*{1}*''), reason code (''*{2}*'').

Explanation
The IBM WebSphere MQ Advanced Message Security interceptor failed to remove the message before it attempted to put it on error handling queue.

Action
Consult the IBM WebSphere MQ documentation for the completion code and reason code, take corrective action. If the problem persists, contact your IBM service representative.

AMS1356
Message too big to fit the buffer, queue (''*{0}*''), MQ completion code (''*{1}*''), reason code (''*{2}*'').

Explanation
The IBM WebSphere MQ Advanced Message Security interceptor failed to unprotect data because provided buffer is too small

Action
Issue MQGET with bigger buffer

amq9001
Channel '*&3*' ended normally.

Explanation
Channel '*&3*' to host '*&5*' ended normally.

Action
None.

amq9002
Channel '*&3*' is starting.

Explanation
Channel '*&3*' is starting.

Action
None.

amq9005

The WebSphere MQ security policy interceptor failed to access the Public-Key Cryptography Standards (PKCS) #11 hardware token.

Explanation

The WebSphere MQ security policy interceptor failed in an attempt to open a Public-Key Cryptography Standards (PKCS) #11 token. Check GSKit ACME GSS minor reason*&1* for '*&3*'.

Action

Make sure that the PKCS #11 token is present and configured correctly, and retry the operation. Verify that the token label, PIN, and library name are configured correctly.

amq9006

The WebSphere MQ security policy interceptor did not attempt to open a PKCS #11 token because it did not have all required configuration information.

Explanation

The WebSphere MQ security policy interceptor did not attempt to open a Public-Key Cryptography Standards (PKCS) #11 token because one or more of the token label, PIN, or shared library name were not configured.

Action

If you want to use a PKCS #11 token, make sure that the token label, PIN, and library name are configured correctly.

amq9007

The WebSphere MQ security policy interceptor failed to convert a Public-Key Cryptography Standards (PKCS) #11 key certificate label.

Explanation

The WebSphere MQ security policy interceptor failed to convert a Public-Key Cryptography Standards (PKCS) #11 key certificate label needed to identify a key certificate item stored in a PKCS #11 token. Check GSKit ACME GSS minor reason*&1*.

Action

Make sure that the PKCS #11 key certificate label is defined correctly.

amq9008

Cannot acquire the certificate for the label:*&3* in the keystore file*&4*. GSKit ACME GSS minor reason is*&1*.

Explanation

WebSphere MQ security policy interceptor was unable to read the certificate for the given label from keystore.

Action

Make sure the label is correctly set as the cms.certificate entry of the configuration file. Check if the keystore contains the certificate for the given label.

amq9009

Cannot acquire credentials. GSKit ACME GSS minor reason is*&1*.

Explanation

WebSphere MQ security policy interceptor was unable to acquire credentials.

Action

Review the configuration to make sure the keystore database and stash files are not broken.

amq9010

WebSphere MQ security policy internal error: message could not be protected because specified encryption algorithm is not valid*&1*.

Explanation

The value identifier is specified to unexpected value.

Action

This is an internal error. Contact your IBM service representative.

amq9011

The WebSphere MQ security policy interceptor failed to turn on the Public-Key Cryptography Standards (PKCS) #11 hardware RSA private key algorithm for this ACME environment. Check GSKit ACME GSS minor reason*&1*.

Explanation

The WebSphere MQ security policy interceptor failed to register the Public-Key Cryptography Standards (PKCS) #11 cryptographic algorithm with the ACME environment.

Action

Make sure that the PKCS #11 token is functioning properly and retry the operation. If the problem persists, contact your IBM service representative.

amq9012

The WebSphere MQ security policy interceptor could not acquire the public key credential.

Explanation

The WebSphere MQ security policy interceptor could not perform a public key infrastructure (PKI) login.

Action

Check the error messages related to acquiring public key credentials to determine the cause of the failure. Check whether user has the permission to read the kdb and stash files and verify whether the kdb file contains a certificate with the label specified. Finally, check whether the certificate has not expired.

amq9013

WebSphere MQ security policy internal error: the Independent Data Unit Protection (IDUP) environment could not be terminated. GSKit reason code*&1*.

Explanation

The WebSphere MQ security policy interceptor could not release the GSKit IDUP environment because an internal error occurred.

Action

Consult the GSKit appendix in the product documentation for the explanation of the GSKit reason code and take corrective action. If the problem persists, contact your IBM service representative.

amq9014

The WebSphere MQ security policy interceptor failed to close a Public-Key Cryptography Standards (PKCS) #11 token. Check GSKit ACME GSS minor reason*&1*.

Explanation

The WebSphere MQ security policy interceptor failed to close a Public-Key Cryptography Standards (PKCS) #11 token.

Action

Make sure that the PKCS #11 token is functioning properly and retry the operation. If the problem persists, contact your IBM service representative.

amq9015

WebSphere MQ security policy internal warning: GSKit could not release*&3*. GSKit reason code*&1*.

Explanation

The WebSphere MQ security policy GSKit call with the indicated reason code failed because it could not release resources back to the system.

Action

No action is required. If the problem persists, contact your IBM service representative.

amq9016

WebSphere MQ security policy internal error: GSKit could not allocate*&3*. GSKit reason code*&1*.

Explanation

The WebSphere MQ security policy GSKit call with the indicated reason code failed because the system could not allocate resources.

Action

Make sure the system meets hardware and software requirements necessary to execute the application, then restart the application.

amq9017

WebSphere MQ security policy internal error: message could not be unprotected: GSKit error code*&1*, reason*&2*.

Explanation

The WebSphere MQ security policy interceptor could not verify or decrypt a message because the indicated GSKit error occurred. This can happen for several reasons, all of which are internal failures: (1) the message is not a valid PKCS#7 message; (2) the sender's certificate does not have the required key usage bit to be able to encrypt the message; (3) the sender's certificate was not recognized as a trusted certificate; (4) receiver is not among the recipients of the message.

Action

Consult the GSKit information in the product documentation for the explanation of the GSKit reason code and take corrective action. If the problem persists, contact your IBM service representative.

amq9018

The specified SHA-2 algorithm '*&3*' is not supported on this platform.

Explanation

The WebSphere MQ security policy interceptor failed to apply a policy as this platform lacks support for SHA-2 signing algorithm.

Action

Check that all platforms that open a queue with a policy that specifies a SHA-2 signing algorithm have the required cryptographic library support.

amq9019

WebSphere MQ security policy internal error: message could not be protected because specified signature algorithm is not valid*&1*

Explanation

The value identifier is specified to unexpected value.

Action

This is an internal error. Contact your IBM service representative.

amq9020

WebSphere MQ security policy internal error: message could not be protected because no recipients' DN is specified.

Explanation

The policy is set to privacy but does not contain any recipient DN.

Action

This is an internal error. Contact your IBM service representative.

amq9021

An error occurred during the certificate import for the following DN:*&3*, result:*&1*

Explanation
The distinguished name is not present in the keystore or invalid.

Action
Consult the GSKit appendix in the product documentation for the explanation of the GSKit reason code and take corrective action. If the problem persists, contact your IBM service representative.

amq9022
An error occurred during the certificate import for the following DN:*&3*, result:*&1*, reason:*&2*.

Explanation
The distinguished name is not present in the keystore or invalid.

Action
Consult the GSKit appendix in the product documentation for the explanation of the GSKit reason code and take corrective action. If the problem persists, contact your IBM service representative.

amq9023
The name of the keystore file '*&3*' was incorrectly provided with the file extension '*&4*'

Explanation
The WebSphere MQ security policy interceptor was unable to find the keystore file. It seems the value of the keystore configuration entry incorrectly contains the file extension.

Action
Ensure the keystore file name specified in the configuration file does not contain a file extension.

amq9024
The keystore file '*&3*' does not exist and the keystore configuration entry incorrectly ends with '*&4*'. Make sure the value of the keystore configuration does not contain the file extension and it points to an existing file.

Explanation
The WebSphere MQ security policy interceptor was unable to find the keystore file. The value of the keystore configuration entry incorrectly contains the file extension of '*&4*' and the resultant filename of '*&3*' does not exist.

Action
Make sure the value of the keystore configuration does not contain the file extension and it points to an existing file.

amq9025
The keystore file*&3&4* does not exist. Make sure the value of the keystore configuration entry points to an existing file.

Explanation
WebSphere MQ security policy interceptor was unable to find the keystore database file.

Action
Make sure the value of the keystore configuration entry points to an existing file.

amq9026
Cannot read the keystore file*&3&4*. Check the file permissions.

Explanation
WebSphere MQ security policy interceptor was unable to read the keystore database file.

Action
Set the proper permissions for the keystore database file.

amq9027
Cannot access the keystore file*&3&4*. Error code*&1*.

Explanation
WebSphere MQ security policy interceptor was unable to open the keystore database file.

Action
Ensure the application accessing the keystore file has appropriate permissions to access the keystore file.

amq9028
The keystore stash file*&3&4* does not exist.

Explanation
WebSphere MQ security policy interceptor was unable to find the keystore stash file.

Action
Ensure the application accessing the keystore stash file has appropriate permissions to access the file.

amq9029
Cannot read the keystore stash file*&3&4*.

Explanation
WebSphere MQ security policy interceptor was unable to read the keystore stash file.

Action
Check the permissions for the keystore stash file.

amq9030
WebSphere MQ security policy internal error: queue information could not be resolved from the current queue object handle (*&1*).

Explanation
The WebSphere MQ security policy interceptor could not resolve queue information from the current queue object handle because the object handle is invalid or unrecognised.

Action
Make sure that the queue has not already been opened by another WebSphere MQ application and that it has not been previously closed. If the problem persists, contact your IBM service representative.

amq9031
WebSphere MQ security policy interceptor has detected an error prior to callback exit execution. WebSphere MQ compcode*&1* : reason*&2*.

Explanation
The WebSphere MQ security policy interceptor received an WebSphere MQ completion code indicating an error prior to interceptor execution.

Action
Consult the product documentation for the completion code and reason code, take corrective action. If the problem persists, contact your IBM service representative.

amq9032
WebSphere MQ security policy interceptor could not find queue manager CodedCharSetId (*&1*). Make sure you have permission to inquire about queue manager properties.

Explanation
The WebSphere MQ security policy interceptor could not find the local queue manager's CodedCharSetId because an internal error occurred.

Action
Make sure that the application has inquire permission to the queue manager. If the problem persists, contact your IBM service representative.

amq9033

WebSphere MQ security policy internal error: could not find local queue manager CodedCharSetId (*&1*).

Explanation

The WebSphere MQ security policy interceptor could not find the local queue manager's CodedCharSetId because an internal error occurred.

Action

If the problem persists, contact your IBM service representative.

amq9034

Message does not have a valid protection type.

Explanation

The WebSphere MQ security policy interceptor detected an invalid protection type in a message header. This usually occurs because the WebSphere MQ message header is not valid.

Action

Retry the operation. If the problem persists, contact your IBM service representative.

amq9035

Message signer is not in the list of authorised signers.

Explanation

The WebSphere MQ security policy interceptor detected that the message is signed by an unauthorised party.

Action

Establish whether the identity associated with the sender of the message is authorized to send messages to this application. Ensure the sender is named in the list of allowed signers on the security policy for the queue.

amq9036

MQOPEN() call failed: reason code (*&1*).

Explanation

The WebSphere MQ security policy interceptor call to the WebSphere MQ MQOPEN() call failed with the indicated error.

Action

Consult the product documentation for an explanation of the error code and suggested corrective action. Ensure that the queue manager is operational and the queue exists.

amq9037

The WebSphere MQ security policy interceptor failed to process a message on queue*&3* with CompCode*&1* Reason code*&2*

Explanation

An unexpected error was encountered whilst applying a security policy to queue*&3*.

Action

This is an internal error. Contact your IBM service representative.

amq9038

The WebSphere MQ security policy interceptor failed to convert the*&3* field of the WebSphere MQ header from CCSID*&1* to CCSID*&2*. Verify that default data conversion has been enabled in WebSphere MQ.

Explanation

The WebSphere MQ security policy interceptor internal error: data conversion failed. This is usually a problem with incompatible character sets.

Action

Enable default data conversion in WebSphere MQ. If the problem persists, contact your IBM service representative.

amq9039

&3 - message was signed by*&4* using*&5*.

Explanation

This is an informational message used to indicate that a message was signed, and to display the signer's name, message timestamp and signature algorithm used.

Action

None.

amq9040

&3 - message was signed and encrypted by*&4* using*&5*.

Explanation

This is an informational message used to indicate that a message was signed and encrypted, and to display the signer's name and encryption algorithm name.

Action

None.

amq9041

Message was not protected.

Explanation

This is an informational message used to indicate that a message was neither signed nor encrypted.

Action

None.

amq9042

WebSphere MQ security policy internal error: unprotected message size*&1* bytes does not match the original message size*&2* bytes.

Explanation

After unprotecting, the message size does not match the original message size. The message header might have been corrupted or tampered with.

Action

Check the message which has been put on the SYSTEM.PROTECTION.ERROR.QUEUE queue to find the cause of the error and the origin of the message. If the problem persists, contact your IBM service representative.

amq9043

Message protection algorithm*&3* is different than the required*&4*.

Explanation

The WebSphere MQ security policy interceptor detected that a message did not meet the encryption strength required by the queue. This usually occurs when encryption strength for a queue is changed while there were still messages in the queue.

Action

Make sure that the same encryption strength is used for the message MQPUT and MQGET. When the queue encryption strength is changed, remove all messages in the queue before resuming normal operation.

amq9044

The WebSphere MQ security policy interceptor has put a defective message on error handling queue*&3*.

Explanation

This is an informational message that indicates the WebSphere MQ security policy put a message it could not interpret on the specified error handling queue.

Action

Make sure only valid messages are put onto queues protected by WebSphere MQ security policies.

amq9045

The WebSphere MQ security policy interceptor failed to put a defective message on error handling queue. WebSphere MQ reason code (*&1*)

Explanation

The WebSphere MQ security policy interceptor was unable to put a message it could not interpret on the error handling queue because the indicated WebSphere MQ error occurred.

Action

Consult the product documentation for more information about the reason code. If the problem persists, contact your IBM service representative.

amq9046

The 'PDMQ' format header from the current message is invalid.

Explanation

The WebSphere MQ security policy interceptor did not find a valid WebSphere MQ security policy header in the the current message.

Action

If the QoP setting for the queue is not set to 'none', make sure that no unprotected messages are routed to this queue.

amq9047

WebSphere MQ security policy found a valid 'PDMQ' format header in the current message.

Explanation

WebSphere MQ security policy interceptor found a valid WebSphere MQ security policy header in the current message.

Action

None.

amq9048

Message quality of protection (*&1*) does not match the quality of protection (*&2*) set for the queue*&3*.

Explanation

The WebSphere MQ security policy interceptor detected a quality of protection (QoP) mismatch between the queue and a message in the queue. QoP mismatches usually occur because the queue QoP is changed between the message put and get.

Action

Make sure that the same queue quality of protection is used for the message put and get. When the queue QoP is changed, clean up all messages in the queue before resuming normal operation.

amq9049

Message with quality of protection (*&1*) higher than (*&2*) set for the queue*&3* has been accepted.

Explanation

The WebSphere MQ security policy interceptor detected a quality of protection (QoP) mismatch between the queue and a message in the queue. QoP mismatch usually occurs when the QoP of the queue is changed between the message put and get.

Action

Make sure that the same queue quality of protection is used for the message put and get. When the QoP of the queue is changed, remove all messages from the queue before resuming any operation.

amq9050

WebSphere MQ security policy could not access the security policy definitions. Major code*&1* : Minor code*&2*

Explanation

The security policy definitions cannot be accessed.

Action

The security policy definitions must be accessible to this application. Check the object authority manager access control for this application to access the SYSTEM.PROTECTION.POLICY.QUEUE.

amq9051

WebSphere MQ could not find the security policy definition. Compcode*&1* : reason*&2*

Explanation

The security policy definition is not defined.

Action

Security policy definition must be defined before this action.

amq9052

Message with no protection has been put into the queue*&3*.

Explanation

The WebSphere MQ security policy interceptor has successfully put a message with a QoP of 'none' onto the selected queue.

Action

None.

amq9053

Message with integrity protection has been put into the queue*&3*.

Explanation

The WebSphere MQ security policy interceptor has successfully put a message with a QoP of 'integrity' onto the selected queue.

Action

None.

amq9054

Message with privacy protection has been put into the queue*&3*.

Explanation

The WebSphere MQ security policy interceptor has successfully put a message with a QoP of 'privacy' onto the selected queue.

Action

None.

amq9055

Quality of protection (QoP)*&1* for queue*&3* is invalid

Explanation

The WebSphere MQ security policy interceptor detected that the quality of protection specified in the security policy for the queue is invalid.

Action

Ensure that the encryption and signature algorithms specified for the queue in the WebSphere MQ security policy definition have valid values.

amq9056

WebSphere MQ security policy internal error: message could not be protected because the specified signature algorithm*&1* is not valid.

Explanation

An unexpected signature algorithm has been specified.

Action

This is an internal error. Contact your IBM service representative.

amq9057

WebSphere MQ security policy internal error: message could not be processed because the specified encryption algorithm*&1* is not valid.

Explanation

An unexpected encryption algorithm has been specified.

Action

This is an internal error. Contact your IBM service representative.

amq9058

The WebSphere MQ security policy interceptor cannot inquire the attributes for queue manager*&3*.

Explanation

The WebSphere MQ security policy interceptor failed to inquire queue manager attributes, compcode*&1*, reason code*&2*.

Action

Make sure that the application has appropriate access control permissions to inquire the queue manager object.

amq9059

The WebSphere MQ security policy interceptor failed to generate a configuration event for queue manager*&3*.

Explanation

The WebSphere MQ security policy interceptor failed to generate a configuration event, comp code*&1*, reason code*&2*.

Action

Ensure that the SYSTEM.ADMIN.CONFIG.EVENT queue is available for output from this process.

amq9060

The WebSphere MQ security policy keystore configuration file contains duplicate key:*&3*.

Explanation

The WebSphere MQ security policy keystore configuration file contains duplicate key:*&3*.

Action

Make sure the keystore configuration file contains all required keys and does not contain duplicate keys.

amq9061

The WebSphere MQ security policy keystore configuration file does not contain key*&3*.

Explanation

The WebSphere MQ security policy keystore configuration file does not contain key*&3*.

Action

Make sure the keystore configuration file contains all required keys and does not contain duplicate keys.

amq9062
The WebSphere MQ security policy interceptor could not read the keystore configuration file:*&3*.

Explanation
The WebSphere MQ security policy interceptor could not read the keystore configuration file:*&3*.

Action
Make sure that the user who executes the WebSphere MQ application has permissions to read the configuration file. Check if the configuration file is not corrupted or empty. If the problem persists, contact your local IBM service representative.

amq9063
The WebSphere MQ security policy interceptor could not parse the keystore configuration file.

Explanation
The WebSphere MQ security policy interceptor could not parse the keystore configuration file.

Action
Make sure the keystore configuration file contains all required keys and does not contain duplicate keys.

amq9064
The WebSphere MQ security policy interceptor failed to enable OCSP checking.

Explanation
The WebSphere MQ security policy interceptor encountered an error while configuring OCSP checking.

Action
Check the OCSP configuration, making sure that all parameters are specified correctly.

amq9065
The WebSphere MQ security policy interceptor failed to process the OCSP configuration.

Explanation
The OCSP configuration for WebSphere MQ is incorrect.

Action
Check the OCSP configuration, making sure that all mandatory parameters are specified.

amq9066
The WebSphere MQ security policy interceptor failed to enable CRL checking.

Explanation
The WebSphere MQ security policy interceptor encountered an error while configuring CRL checking.

Action
Check the CRL configuration, making sure that all parameters are correctly specified.

amq9067
The WebSphere MQ security policy interceptor failed to process the CRL configuration.

Explanation
The WebSphere MQ security policy interceptor failedto process the CRL configuration information.

Action
Check the CRL configuration, making sure that all mandatory parameters are specified.

amq9068
The WebSphere MQ security policy interceptor could not enable revocation checking.

Explanation

The WebSphere MQ security policy interceptor could not enable OCSP or CRL because the required functionality is not supported by the version of GSKit that is currently being used.

Action

The currently enabled version of GSKit does not meet the minimum requirements for WebSphere MQ security policy revocation checking. Install a newer version of GSKit that does meet the minimum requirements.

amq9069

The WebSphere MQ security policy interceptor failed to validate a certificate, GSKit status:*&3*.

Explanation

The WebSphere MQ security policy interceptor failed to validate a certificate.

Action

Read the GSKit log to determine the cause of validation failure.

amq9070

The WebSphere MQ security policy interceptor failed to validate a certificate.

Explanation

The WebSphere MQ security policy interceptor could not validate a certificate.

Action

Read the GSKit logs to determine the cause of validation failure.

amq9071

WebSphere MQ security policy interceptor internal error: could not read GSKit attribute*&3*, GSKit reason code:*&1*.

Explanation

The GSKit gskacme_cred_get function failed.

Action

Make sure you are using the correct version of GSKit for this installation of WebSphere MQ.

amq9072

The certificate revocation status is UNKNOWN, GSKit log:*&3*.

Explanation

The WebSphere MQ security policy interceptor failed to determine the certificate revocation status.

Action

Read the GSKit log to find out the certificate DN and serial number.

amq9073

The certificate revocation status is UNKNOWN, GSKit log:*&3*.

Explanation

The WebSphere MQ security policy interceptor failed to determine the certificate revocation status.

Action

Read the GSKit log to find out the certificate DN and serial number.

amq9074

The Trial Period license for this copy of WebSphere MQ Advanced Message Security has expired.

Explanation

This copy of WebSphere MQ Advanced Message Security was licensed to be used in trial mode for a limited period only. This period has expired.

Action
Install a Production license for this copy of WebSphere MQ Advanced Message Security.

amq9075
The Beta license for this copy of WebSphere MQ Advanced Message Security has expired.

Explanation
This copy of WebSphere MQ Advanced Message Security was licensed to be used for Beta testing for a limited period only. This period has expired.

Action
Install a Production license for this copy of WebSphere MQ Advanced Message Security.

amq9076
There are*&1* days left in the trial period for this copy of WebSphere MQ Advanced Message Security.

Explanation
This copy of WebSphere MQ Advanced Message Security is licensed for a limited period only.

Action
None.

amq9077
This is the final day of the trial period for this copy of WebSphere MQ Advanced Message Security.

Explanation
This copy of WebSphere MQ Advanced Message Security is licensed for a limited period only.

Action
Install a Production license for this copy of WebSphere MQ Advanced Message Security.

amq9078
There is one day left in the trial period for this copy of WebSphere MQ Advanced Message Security.

Explanation
This copy of WebSphere MQ Advanced Message Security is licensed for a limited period only.

Action
None.

amq9079
There are*&1* days left in the beta test period for this copy of WebSphere MQ Advanced Message Security.

Explanation
This copy of WebSphere MQ Advanced Message Security is licensed for a limited period only.

Action
None.

amq9080
There is one day left in the Beta test period for this copy of WebSphere MQ Advanced Message Security.

Explanation
This copy of WebSphere MQ Advanced Message Security is licensed for a limited period only.

Action
None.

amq9081

This is the final day of the Beta test period for this copy of WebSphere MQ Advanced Message Security.

Explanation

This copy of WebSphere MQ Advanced Message Security is licensed for a limited period only.

Action

Install a Production license for this copy of WebSphere MQ Advanced Message Security.

amq9082

No policies found.

Explanation

There are no policies defined.

Action

None.

Index

Special characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com®, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

This product includes software developed by the Eclipse Project (http://www.eclipse.org/).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Sending your comments to IBM

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

- Send an email to ibmkc@us.ibm.com
- Use the form on the web here: www.ibm.com/software/data/rcf/

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

Include the following information:

- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number
- The topic and page number related to your comment
- The text of your comment

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

Thank you for your participation.